1. La instalación es siguiente, siguiente y el ejecutable se descarga desde <https://www.python.org/downloads/>.
2. Se puede ejecutar con Sublime Text, desde la consola de Windows, desde la consola de Python y desde <https://jupyter.org/try>.
3. **Variables**:

Ejemplo:

Numero = 5

Cadena = “hola mundo”

* Cuando son cadenas de caracteres se coloca en comillas simples.
* Comienza con letra o guion bajo.
* Es sensible a mayúscula.
* Se concatena con el +.
* Para saber el tipo de la variable se usa la función **type()**.
* Los decimales se dividen por coma.
* Para convertir una variable a string se usa la función **str()**.
* Para convertir una variable a entero se usa la función **int()**.
* Para convertir una variable a decimal se usa la función **float()**.
* Es necesario convertir un número a string si se quiere concatenar con una cadena.
* Los índices comienzan en 0 y -1 si se quiere de forma inversa. Ejemplo en la variable Cadena anterior Cadena [5] sería la letra la “m” y Cadena [-1] es “o”.
* Para mostrar solo parte de una cadena se hace, por ejemplo, Cadena [2:7] que es “la mu”, ya que es posición inicial: posición final menos 1., otro ejemplo Cadena [2:] que toma desde la posición 2 al final.
* Para saber la longitud de una cadena se usa la función **len()**.
* Para mostrar una cadena a mayúscula se usa la función **variable.upper()**. Se muestra, pero la variable va a continuar siendo minúscula.
* Similar con **variable.lower()**.
* Para crear una lista con las palabras de una cadena se usa **cadena.split()**, también puede ser, por ejemplo: **cadena.split(‘,’)**.
* En un **print** la coma es un espacio.
* Para imprimir se usa **print()**, por ejemplo

**print("Buenos días " + nombre)**

* Otra forma de imprimir es con la función **.format()**, por ejemplo,

**print("Buenos días {}, feliz {} cumpleaños.".format(nombre,edad))**

* Para formatear un número y decirle que solo muestre 3 decimales se puede hacer, por ejemplo:

**resultado = 10/3**

**print("El resultado es {r:1.3f}".format(r=resultado))**

* Otra forma de imprimir.

**print(f"Buenos días {nombre}, feliz {edad} cumpleaños.")**

1. **Tomar texto por pantalla**

* Se usa la función **input()**. Ejemplo,

**print("Introduzca un nombre: ")**

**entrada = input()**

**print("Hola " + entrada)**

1. **Operadores**

* Aritméticos (+, -, \*, /, %, \*\*, //).
* Asignación (=, +=, -=, \*=, /=, \*\*=).
* Comparación (==, !=, >, <, >=, <=).
* Lógicos (and, or, not).
* Identidad (is, is not), verifica si 2 objetos son iguales o no.
* Pertenencia (in, not in), verifica si un elemento esta dentro de un objeto.

1. **Colección de datos**

* Listas:
  + La forma es, por ejemplo, colores = [“rojo”, “amarillo”, “verde”].
  + Se puede modificar con su índice, que comienza en 0, por ejemplo, colores[0] = “azul”.
  + Con la función **len()** se calcula el largo de la lista.
  + Con la función **append()** se puede agregar un nuevo elemento a la lista, por ejemplo, colores.append(“naranja”).
  + Con la función **remove()** se puede eliminar un elemento, por ejemplo, colores.remove(“rojo”).
  + Para vaciar una lista se usa la función **clear()**, por ejemplo colores.clear().
* Tuplas:
  + Colección de elementos ordenados que no se pueden eliminar.
  + Se coloca entre paréntesis, por ejemplo, tupla\_colores = (“rojo, “verde”, “amarillo”).
  + Se trabaja igual que una lista, pero no se puede modificar, eliminar, ni agregar nuevos elementos.
* Conjuntos:
  + Colección de elementos desordenados, es decir, no posee índice.
  + Se identifica con llaves, por ejemplo, conjunto\_colores = {“rojo”, “verde”, “azul”}.
  + Con la función **len()** se puede saber su tamaño.
  + Se puede agregar elementos, por ejemplo, conjunto\_colores.**add**(“amarillo”).
  + Se puede remover un elemento, por ejemplo, conjunto\_colores.**remove**(“verde”).
* Diccionarios:
  + Colección de elementos indexados y no ordenados.
  + Se escribe entre llaves y posee clave valor, por ejemplo, diccionario\_colores = {“red”:”rojo”, “blue”:”azul”, “yellow”:”amarillo”}.
  + Se accede a cada elemento con su clave, por ejemplo, diccionario\_colores[“red”].
  + Para agregar un nuevo valor, por ejemplo, diccionario\_colores[“black”] = “negro”.
  + Para eliminar un valor se hace, por ejemplo, diccionario\_colores.**pop**(“yellow”).
  + Otra forma de eliminar es con la función **del()**, por ejemplo, del(diccionario\_colores[“black”]).
  + Para imprimir clave y valor se hace, por ejemplo,

**For clave,valor in diccionario\_colores.items():**

**Print(clave,valor)**

1. **Bucles y condiciones**

* Condiciones if, elif, else:

**If(condiciones1):**

**<operaciones1>**

**Elif(condiciones2):**

**<operaciones2>**

**Else:**

**<operaciones3>**

* Blucle for:
  + Ejemplos:

colores = ["rojo", "verde", "azul"]

for color in colores:

print(color)

cadena = "Hola mundo"

for caracter in cadena:

print(caracter)

for numero in range(8):

print(numero)

for numero in range(3,8):

print(numero)

for numero in range(3,8,2):

print(numero)

for numero in range(3,8,4):

print(numero)

for numero in range(10):

if(numero == 5):

break

print(numero)

for numero1 in range(4):

for numero2 in range(3):

print(numero1, numero2)

* + Para generar un rango de valores se usa la función **range(N°)**, por ejemplo si quiero un rango que va del 0 al 7 se hace range(8), y para recorrerlo:

For numero in range(8):

Print(numero)

* + El **range** también puede ser de esta forma **range(3,8)**, generando un rango del 3 al 7.
  + Para que recorra el **range**, por ejemplo del 3 al 7 de dos en dos se hace **range(3,8,2)**.
  + El **break** también funciona para parar las iteraciones.
  + El **continue** también funciona para parar la iteración actual.
* Bucle while:
  + Por ejemplo:

valor = 1

fin = 10

while(valor < fin):

print(valor)

valor +=1

* + También funciona el **break** y el **continue**.

1. **Clases, objetos y funciones**

* Clases y objetos:
  + Una clase es como un constructor de objetos.
  + Un objeto es la instancia a una clase.

**class ClaseSilla:**

**color = 'blanco'**

**precio = 100**

**objetoSilla1 = ClaseSilla()**

**objetoSilla1.color**

**objetoSilla1.precio**

* + Otro ejemplo:

**class Persona:**

**# Método constructor, donde self indica que ocupa**

**# las propiedades de la clase**

**def \_\_init\_\_(self, nombre, edad):**

**self.nombre = nombre**

**self.edad = edad**

**# Método que devuelve un saludo**

**def saludar(self):**

**print('Hola, me llamo {} y tengo {} años.'.format(self.nombre, self.edad))**

**# Se instancia la clase**

**persona1 = Persona('Juan',33)**

* + Otro ejemplo:

**class Coche:**

**def \_\_init\_\_(self, marca, color, combustible, cilindrada):**

**self.marca = marca**

**self.color = color**

**self.combustible = combustible**

**self.cilindrada = cilindrada**

**def mostrar\_caracteristicas(self):**

**print("{}, {}, {}, {}".format(self.marca, self.color, self.combustible, self.cilindrada))**

**coche1 = Coche('Opel', 'Rojo', 'Gasolina', '1.6')**

**coche1.mostrar\_caracteristicas()**

* Funciones
  + Es un código que se ejecuta cuando es llamado.
  + Se declara con la variable **def**.
  + Se llama usando el nombre entre paréntesis. Por ejemplo:

**def saludar():**

**print("Buenos días")**

**saludar()**

* También se le puede pasar parámetros. Por ejemplo:

**def saludar(nombre):**

**print("Buenos días {}".format(nombre))**

**nombre = 'Antonio'**

**saludar(nombre)**

* Primero se declaran las funciones y/o clase y luego se usan. Ejemplo:

**numero1 = 10**

**numero2 = 20**

**def suma(numero1, numero2):**

**suma = numero1 + numero2**

**return suma**

**resultado = suma(numero1, numero2)**

**print("El resultado de la suma de {} y {} es {}".format(numero1, numero2, resultado))**

* Para pasar por referencia se hace, por ejemplo:

**colores = ['rojo', 'verde', 'azul']**

**def anadirColor(colores, color):**

**colores.append(color)**

**color = 'negro'**

**anadirColor(colores, color)**

* Funciones lambda
  + Es una función pequeña que entrega un resultado.
  + Por ejemplo,

**resultado = lambda numero : numero + 30**

**resultado(10)**

* + Ejemplo 2,

**resultado2 = lambda numero1, numero2 : numero1 + numero2**

**resultado2(10,30)**

* + Ejemplo 3,

**media = lambda nota1, nota2, nota3 : (nota1 + nota2 + nota3) / 3**

**media(10, 20, 30)**

1. **Módulos**

* Es un fichero que contiene un conjunto de instrucciones que se pueden usar.
* Por ejemplo:

# Dentro de un archivo **modulo.py** colocamos:

**def saludar(nombre):**

**print("Hola, soy " + nombre)**

**def despedirse(nombre):**

**print(“Adiós “ + nombre**

#En otro archivo se llama al módulo, por ejemplo, colocamos:

**import modulo**

**modulo.saludar('Juan')**

* Si quiero solo usar una sola función de un módulo, se hace, por ejemplo:

**from modulo import despedirse**

#En este caso se usa directo la función y no se necesita anteponer modulo.

**despedirse('Pedro')**

* Se puede usar alias para utilizar las funciones que están en el módulo con otro nombre, por ejemplo:

**from modulo import despedirse as adios**

**adios('Pedro')**

* Para instalar módulos nuevos se usa **Pip** que es un gestor de paquetes y módulos para phyton. Dentro de la consola se puede hacer:
  + Versión:

**# pip --version**

* + Listar módulos:

**# pip list**

* + Instalar un módulo:

**# pip Install <modulo>**

* + Ejemplo con el modulo camelcase que permite pasar a mayúscula:

**# pip**

**import camelcase**

**camel = camelcase.CamelCase()**

**texto = "mi nombre es antonio"**

**print(camel.hump(texto))**

El resultado es **Mi Nombre Es Antonio**

* + Desinstalar un módulo:

**# pip uninstrall camelcase**

* + Ejemplo:

#Modulo1

**class Coche:**

**def \_\_init\_\_(self, marca, color, combustible, cilindrada):**

**self.marca = marca**

**self.color = color**

**self.combustible = combustible**

**self.cilindrada = cilindrada**

**def mostrar\_caracteristicas(self):**

**print("{}, {}, {}, {}".format(self.marca, self.color, self.combustible, self.cilindrada))**

**media = lambda nota1, nota2, nota3 : (nota1 + nota2 + nota3) / 3**

#Programa1

**import modulo1**

**coche1 = modulo1.Coche('Opel', 'Rojo', 'Gasolina', '1.6')**

**coche1.mostrar\_caracteristicas()**

**nota1 = 5.0**

**nota2 = 6.1**

**nota3 = 4.5**

**promedioNotas = modulo1.media(nota1, nota2, nota3)**

**print("El promedio entre las nota1 = {}, nota2 = {} y nota3 = {} es {}".format(nota1, nota2, nota3, promedioNotas))**

1. **Ficheros**:

* Leer ficheros:
  + Por ejemplo, para un fichero de texto:

**fichero = open("ficherotexto.txt","rt")**

**datos\_fichero = fichero.read()**

**print(datos\_fichero)**

* Grabar ficheros:
  + Por ejemplo, crear y agregar información a un fichero de texto nuevo:

**fichero = open("fichero\_para\_grabar.txt", "wt")**

**texto\_de\_fichero = "Hola, esta es la línea que vamos a grabar en el fichero de texto"**

**fichero.write(texto\_de\_fichero)**

**fichero.close()**

* Incluir datos de un fichero:
  + Por ejemplo, agregar información a un fichero existente:

**fichero = open("ficherotexto.txt", "at")**

**cadena\_para\_incluir = "\nEsta es la tercera fila del fichero"**

**fichero.write(cadena\_para\_incluir)**

**fichero.close()**

* Borrar un fichero:

**import os**

**os.remove("ficherotexto.txt")**

* Ejemplo:

#moduloficherosdos.py

**class Fichero:**

**def \_\_init\_\_(self, nombre):**

**self.nombre = nombre**

**def grabar\_fichero(self, texto):**

**fichero = open(self.nombre, "wt")**

**fichero.write(texto)**

**fichero.close()**

**def incluir\_fichero(self, texto):**

**fichero = open(self.nombre, "at")**

**fichero.write(texto)**

**fichero.close()**

**def leer\_fichero(self):**

**fichero = open(self.nombre, "rt")**

**texto = fichero.read()**

**return texto**

# programa2.py

**import moduloficherosdos**

**nombre\_fichero = "ficherin.txt"**

**fichero = moduloficherosdos.Fichero(nombre\_fichero)**

**texto = 'Esta es la primera fila del fichero.\nEsta es la segunda fila del fichero.'**

**fichero.grabar\_fichero(texto)**

**texto = '\nEsta es la tercera fila del fichero'**

**fichero.incluir\_fichero(texto)**

**texto\_leido = fichero.leer\_fichero()**

**print(texto\_leido)**

1. **Ficheros binarios**

* Grabar fichero binario:
  + Ejemplo:

i**mport pickle**

**lista\_colores = ["azul", "verde", "rojo", "amarillo"]**

**fichero = open("ficherocolores.pckl", "wb")**

**pickle.dump(lista\_colores, fichero)**

**fichero.close()**

* Leer fichero binario:
  + Ejemplo:

**import pickle**

**fichero = open("ficherocolores.pckl", "rb")**

**lista\_leida\_fichero = pickle.load(fichero)**

**print(lista\_leida\_fichero)**

1. **Gestión de errores**

* Se usa **try**, **except**, **else** y **finally**.
* Ejemplos:

1)

**try:**

**numero1 = 5**

**numero2 = 0**

**division = numero1 / numero2**

**print(division)**

**except:**

**print("Ha habido un error")**

2)

**try:**

**numero1 = 5**

**numero2 = 0**

**division = numero1 / numero2**

**print(division)**

**except ZeroDivisionError:**

**print("Error, no se puede dividir por cero.")**

**except:**

**print("Ha habido un error")**

3)

**try:**

**numero1 = 5**

**numero2 = 1**

**division = numero1 / numero2**

**print(division)**

**except ZeroDivisionError:**

**print("Error, no se puede dividir por cero.")**

**except:**

**print("Ha habido un error")**

**else:**

**print("La división funciono correctamente.")**

4)

**try:**

**numero1 = 5**

**numero2 = 0**

**division = numero1 / numero2**

**print(division)**

**except ZeroDivisionError:**

**print("Error, no se puede dividir por cero.")**

**except:**

**print("Ha habido un error")**

**else:**

**print("La división funciono correctamente.")**

**finally:**

**print("Esta prueba del try se ha acabado")**

5)

**def operaciones(numero1, numero2, numero3):**

**try:**

**resultado = numero1 / (numero2-numero3)**

**print(resultado)**

**except ZeroDivisionError:**

**print("Error, no se puede dividir por cero. Los últimos dos número debe ser diferentes")**

**else:**

**print("El resultado obtenido es correcto.")**

**finally:**

**print("Ejercicio finalizado.")**

**numero1 = 5**

**numero2 = 3**

**numero3 = 3**

**resultado = operaciones(numero1, numero2, numero3)**

**print(resultado)**

1. **Expresiones regulares, JSON, fecha y hora**

* Expresiones regulares:
  + Search: Realiza una búsqueda de un patrón en una cadena.

Ejemplos:

1)

**texto = "Hola, mi nombre es Antonio"**

**import re**

**resultado = re.search("nombre", texto)**

**if(resultado):**

**print("Cadena encontrada: {}".format(resultado))**

**else:**

**print("Cadena NO encontrada")**

2) Con el $ busca al final y con el ^ busca al principio.

**texto = "Hola, mi nombre es Antonio"**

**import re**

**resultado = re.search("Antonio$", texto)**

**if(resultado):**

**print("Cadena encontrada: {}".format(resultado))**

**else:**

**print("Cadena NO encontrada")**

**texto = "Hola, mi nombre es Antonio"**

**import re**

**resultado = re.search("^Hola", texto)**

**if(resultado):**

**print("Cadena encontrada: {}".format(resultado))**

**else:**

**print("Cadena NO encontrada")**

3) Con el .\* se puede indicar que existen mas caracteres entre lo que se esta buscando.

**texto = "Hola, mi nombre es Antonio"**

**import re**

**resultado = re.search("mi.\*es", texto)**

**if(resultado):**

**print("Cadena encontrada: {}".format(resultado))**

**else:**

**print("Cadena NO encontrada")**

4) Con los métodos **start** y **end** se puede obtener la posición del hallazgo. Ejemplo:

**import re**

**def buscar(texto, palabra\_a\_buscar):**

**resultado = re.search(palabra\_a\_buscar, texto)**

**return resultado**

**texto = "Esto es una frase de pruebas para hacer busquedas"**

**palabra\_a\_buscar = "frase"**

**resultado = buscar(texto, palabra\_a\_buscar)**

**if(resultado):**

**posicion\_inicial = resultado.start()**

**posicion\_final = resultado.end()**

**print("Frase encontrada desde la posición {} hasta la {}".format(posicion\_inicial, posicion\_final))**

**else:**

**print("Frase no encontrada")**

* + Findall: Busca todas las ocurrencias en una cadena.

Ejemplos:

1)

**texto = """**

**El coche de luis es rojo,**

**el coche de Antonio es blanco,**

**y el coche de Maria es rojo**

**"""**

**import re**

**print(re.findall("coche.\*rojo", texto))**

* + Split: Divide una cadena a partir de un patrón.

Ejemplos:

1)

**texto = "La silla es blanca y vale 80"**

**import re**

**resultado = re.split("\s", texto)**

**print(resultado)**

* + Sub: Substituye todas las coincidencias de una cadena.

Ejemplos:

1)

**texto = "La silla es blanca y vale 80"**

**import re**

**resultado = re.sub("blanca", "roja", texto)**

**print(resultado)**

* JSON: Forma de almacenar y comparar datos.

**producto1 = {"nombre":"silla", "color":"blanco", "precio":80}**

**import json**

**#Genera el JSON**

**estructura\_json = json.dumps(producto1)**

**print(estructura\_json)**

**#Pasa a JSON a una estructura normal**

**productos2 = json.loads(estructura\_json)**

**print(productos2)**

* Fecha y hora:

**from datetime import datetime**

**fechayhora = datetime.now()**

**print(fechayhora)**

**ano = fechayhora.year**

**mes = fechayhora.month**

**dia = fechayhora.day**

**hora = fechayhora.hour**

**minutos = fechayhora.minute**

**segundos = fechayhora.second**

**microsegundos = fechayhora.microsecond**

**print("La hora es {}:{}:{}".format(hora,minutos,segundos))**

**print("La fecha es {}/{}/{}".format(dia,mes,ano))**

1. **Bases de datos**

* Crear base de datos:

**import sqlite3**

**conexion = sqlite3.connect("basedatos1.db")**

**conexion.close()**

* Crear tabla:
  + El método **cursor** que crea un objeto que permite ejecutar sentencias en la base de datos.
  + El método **execute** permite ejecutar cualquier sentencia SQL.
  + El método **commit** le indica al sistema que la sentencia esta correcta y que la vamos a mantener.

**import sqlite3**

**conexion = sqlite3.connect("basedatos1.db")**

**conexion.cursor()**

**cursor = conexion.cursor()**

**cursor.execute("CREATE TABLE PERSONAS (nombre TEXT, apellido1 TEXT, apellido2 TEXT, edad INTEGER)")**

**conexion.commit()**

**conexion.close()**

* Insertar registro:

**import sqlite3**

**conexion = sqlite3.connect("../basedatos1.db")**

**cursor = conexion.cursor()**

**cursor.execute("INSERT INTO PERSONAS VALUES ('Antonio','Perez','Gomez',35)")**

**conexion.commit()**

**conexion.close()**

* Insertar varios registros:
  + Se usa el método **executemany**.

**import sqlite3**

**conexion = sqlite3.connect("../basedatos1.db")**

**cursor = conexion.cursor()**

**lista\_personas = [**

**('Pedro','Rodriguez','Perez',26),**

**('Maria','Lopez','Gomez',45),**

**('Luis','Gonzalez','Perez',46)**

**]**

**cursor.executemany("INSERT INTO PERSONAS VALUES (?,?,?,?)", lista\_personas)**

**conexion.commit()**

**conexion.close()**

* Consultar registros:
  + Se usa el método **fetchall** que recoge las filas y columnas.

Ejemplo 1)

**import sqlite3**

**conexion = sqlite3.connect("../basedatos1.db")**

**cursor = conexion.cursor()**

**cursor.execute("SELECT \* FROM PERSONAS")**

**personas = cursor.fetchall()**

**for persona in personas:**

**print(persona)**

**conexion.close()**

Ejemplo 2)

**import sqlite3**

**conexion = sqlite3.connect("../basedatos1.db")**

**cursor = conexion.cursor()**

**cursor.execute("SELECT \* FROM PERSONAS WHERE edad > 40")**

**personas\_seleccionadas = cursor.fetchall()**

**for persona in personas\_seleccionadas:**

**print(persona)**

**conexion.close()**

Ejemplo 3)

**import sqlite3**

**conexion = sqlite3.connect("../basedatos1.db")**

**cursor = conexion.cursor()**

**cursor.execute("SELECT \* FROM PERSONAS ORDER BY edad")**

**lista\_personas\_ordenada = cursor.fetchall()**

**for persona in lista\_personas\_ordenada:**

**print(persona)**

**conexion.close()**

* Borrar datos:

**import sqlite3**

**conexion = sqlite3.connect("../basedatos1.db")**

**cursor = conexion.cursor()**

**cursor.execute("DELETE FROM PERSONAS WHERE nombre = 'Luis'")**

**conexion.commit()**

**conexion.close()**

* Actualizar datos:

**import sqlite3**

**conexion = sqlite3.connect("../basedatos1.db")**

**cursor = conexion.cursor()**

**cursor.execute("UPDATE PERSONAS SET edad = 30 WHERE nombre = 'Antonio'")**

**conexion.commit()**

**conexion.close()**

* Ejemplo completo:

**import sqlite3**

**conexion = sqlite3.connect("basededatos.db")**

**cursor = conexion.cursor()**

**cursor.execute("CREATE TABLE PRODUCTOS (id INTEGER, nombre TEXT, precio INTEGER)")**

**conexion.commit()**

**lista\_productos = [**

**(1,'Impresora',300),**

**(2,'Ratón',20),**

**(3,'Ordenador',600)**

**]**

**cursor.executemany("INSERT INTO PRODUCTOS VALUES (?,?,?)", lista\_productos)**

**conexion.commit()**

**cursor.execute("SELECT \* FROM PRODUCTOS")**

**productos = cursor.fetchall()**

**for producto in productos:**

**print(producto)**

**conexion.close()**

1. **Interfaz gráfica con el módulo tkinter**

* Introducción:
  + El módulo **tkinter** tiene una serie de componentes llamados Widgets que son componentes gráficos.
  + **Tk**: componente que contiene el resto de los componentes, es decir, la raíz.
  + **Frame**: es un marco que puede contener otros widgets y que tiene un tamaño propio.
  + **Label**: etiqueta donde se puede insertar texto.
  + **Entry**: campo de texto sencillo para escribir un texto corto, como un nombre en un formulario.
  + **Text**: campo de texto grande para escribir varias líneas de texto, como un campo comentario.
  + **Button**: Botón.
  + **RadioButton**: Botón de opciones.
  + **CheckButton**: Botón de marcado.
  + **Menu**: componente para crear los menús de las aplicaciones.
  + **Dialogs**: componente para crear ventanas emergentes como alertas o confirmaciones.
* Componente raíz (tk):
  + El método **mainloop** provoca que se ejecute todo el rato.

**import tkinter**

**raiz = tkinter.Tk()**

**raiz.title("Mi programa")**

**raiz.mainloop()**

* Componente frame:
  + El método **pack** hace que se muestre por pantalla.
  + **Bg** es el color de fondo, **width** el ancho y **height** el alto.

**import tkinter**

**raiz = tkinter.Tk()**

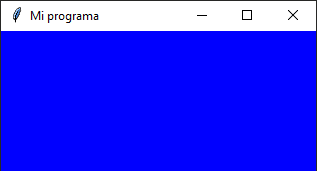
**raiz.title("Mi programa")**

**frame = tkinter.Frame(raiz)**

**frame.config(bg="blue",width=400,height=300)**

**frame.pack()**

**raiz.mainloop()**



* Componente label:
  + **Fg** el color de la letra, **bg** el color del fondo de la letra y **font** es la fuente y tamaño.

**import tkinter**

**raiz = tkinter.Tk()**

**raiz.title("Mi programa")**

**texto = "Hola mundo"**

**etiqueta = tkinter.Label(raiz,text=texto)**

**etiqueta.config(fg="green",bg="lightgrey",font=("Cortana",30))**

**etiqueta.pack()**

**raiz.mainloop()**



* Componente entry:
  + Permite ingresar información por teclado.
  + **Show** permite convertir el texto en un campo de contraseña.

**import tkinter**

**raiz = tkinter.Tk()**

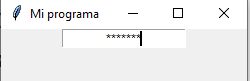
**raiz.title("Mi programa")**

**entrada = tkinter.Entry(raiz)**

**entrada.config(justify="center",show="\*")**

**entrada.pack()**

**raiz.mainloop()**



* Componente text:
  + **Padx** es el espaciado en el eje X, **Pady** es el espaciado en el eje Y y **selectbackground** es el color del seleccionado.

**import tkinter**

**raiz = tkinter.Tk()**

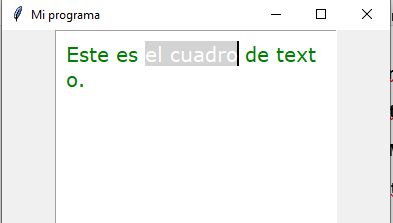
**raiz.title("Mi programa")**

**entrada = tkinter.Text(raiz)**

**entrada.config(width=20,height=10,font=("Verdana",15),padx=10,pady=10,fg="green",selectbackground="lightgrey")**

**entrada.pack()**

**raiz.mainloop()**



* Componente button:

**import tkinter**

**raiz = tkinter.Tk()**

**raiz.title("Mi programa")**

**def accion():**

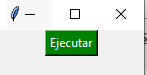
**print("Hola mundo")**

**boton = tkinter.Button(raiz, text="Ejecutar", command=accion)**

**boton.config(fg="white", bg="green")**

**boton.pack()**

**raiz.mainloop()**



* Componente radiobutton:

**import tkinter**

**raiz = tkinter.Tk()**

**raiz.title("Mi programa")**

**def seleccionar():**

**print("La opción seleccionada es {}".format(opcion.get()))**

**opcion = tkinter.IntVar()**

**botonradio1 = tkinter.Radiobutton(raiz, text="Opción 1", variable=opcion, value=1, command=seleccionar)**

**botonradio1.pack()**

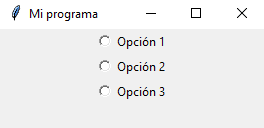
**botonradio2 = tkinter.Radiobutton(raiz, text="Opción 2", variable=opcion, value=2, command=seleccionar)**

**botonradio2.pack()**

**botonradio3 = tkinter.Radiobutton(raiz, text="Opción 3", variable=opcion, value=3, command=seleccionar)**

**botonradio3.pack()**

**raiz.mainloop()**



* Componente checkbutton:

**import tkinter**

**raiz = tkinter.Tk()**

**raiz.title("Mi programa")**

**def verificar():**

**valor = check1.get()**

**if(valor == 1):**

**print("El check esta activado")**

**else:**

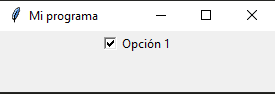
**print("El check esta desactivado")**

**check1 = tkinter.IntVar()**

**boton1 = tkinter.Checkbutton(raiz, text="Opción 1", variable=check1, onvalue=1, offvalue=0, command=verificar)**

**boton1.pack()**

**raiz.mainloop()**



* Componente messagebox:
  + Desde tkinter hay que importar la librería **messagebox**.

**import tkinter**

**from tkinter import messagebox**

**raiz=tkinter.Tk()**

**raiz.title("Mi programa")**

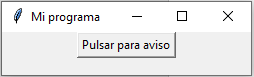
**def avisar():**

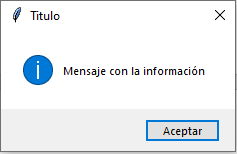
**tkinter.messagebox.showinfo("Titulo", "Mensaje con la información")**

**boton = tkinter.Button(raiz, text="Pulsar para aviso", command=avisar)**

**boton.pack()**

**raiz.mainloop()**





* + Para realizar una consulta con un messagebox:

**import tkinter**

**from tkinter import messagebox**

**raiz = tkinter.Tk()**

**raiz.title("Mi programa")**

**def preguntar():**

**resultado = tkinter.messagebox.askquestion("Titulo de la pregunta", "¿Quieres borrar este fichero?")**

**if(resultado == "yes"):**

**print("Si, quiero borrar el fichero")**

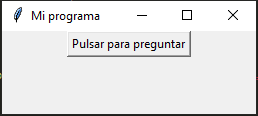
**else:**

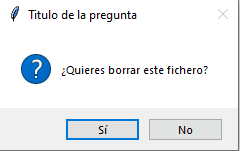
**print("No, no quiero borrar el fichero")**

**boton = tkinter.Button(raiz, text="Pulsar para preguntar", command=preguntar)**

**boton.pack()**

**raiz.mainloop()**





* Componente filedialog:
  + Desde tkinter hay que importar la librería **filedialog**.

**import tkinter**

**from tkinter import filedialog**

**raiz = tkinter.Tk()**

**raiz.title("Mi programa")**

**def abrirfichero():**

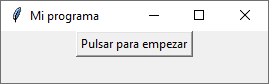
**rutafichero = filedialog.askopenfilename(title="Abrir un fichero")**

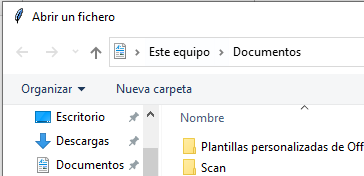
**print(rutafichero)**

**boton = tkinter.Button(raiz, text="Pulsar para empezar", command=abrirfichero)**

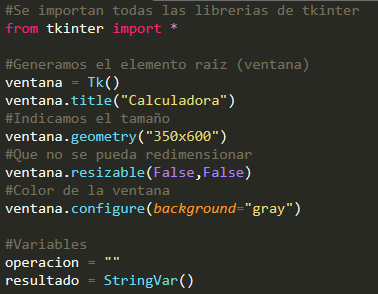
**boton.pack()**

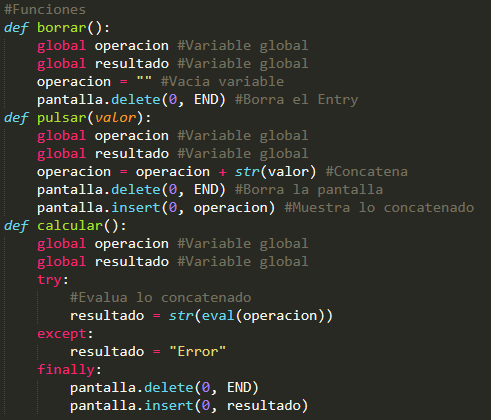
**raiz.mainloop()**

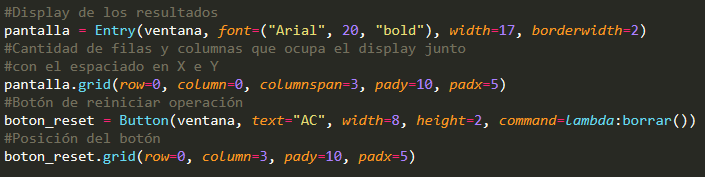


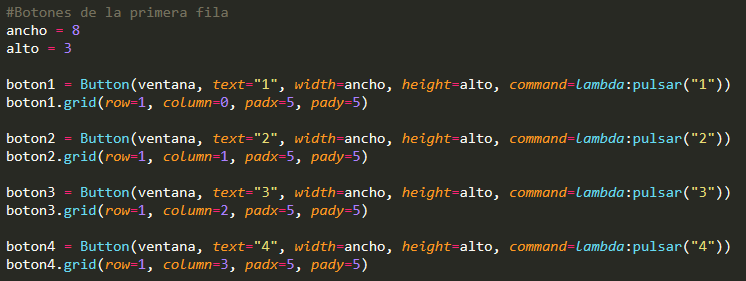


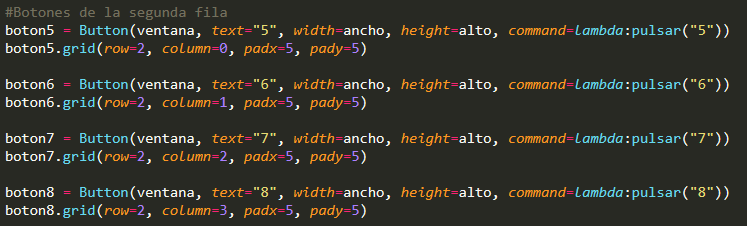
* Ejercicio – Calculadora:

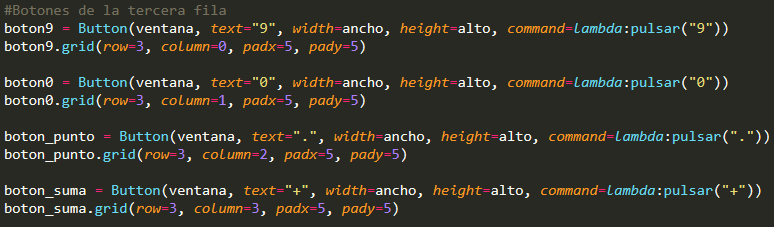


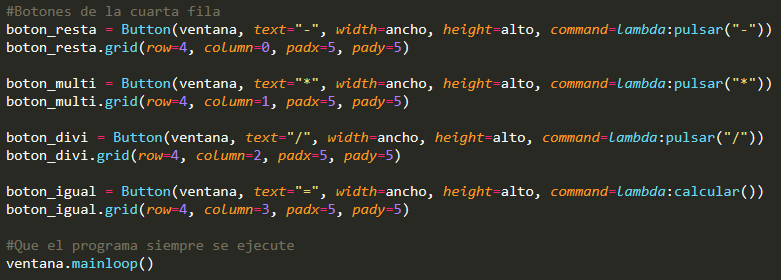






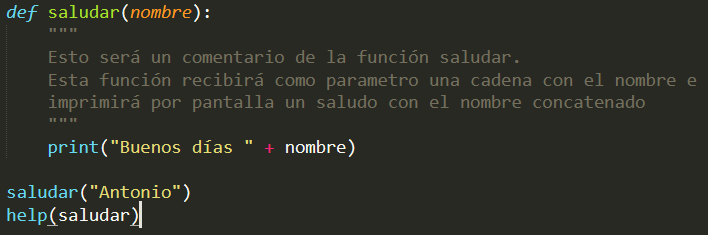




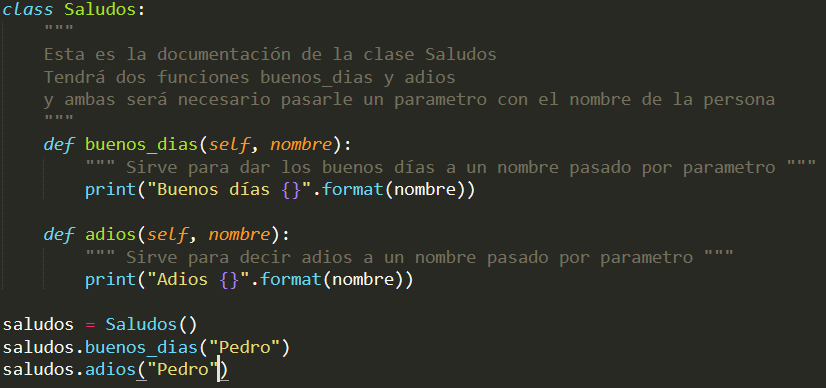


1. **Generar documentación automáticamente**:

* Docstrings: Son cadenas para generar automáticamente documentación del código.
  + Los comentarios comienzan y terminan con 3 comillas dobles.
  + Para mostrar los comentarios se usa **help(nombre\_función)**.



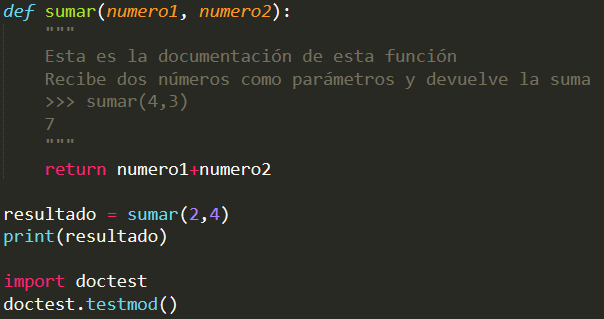
* + Help se puede usar para cualquier función, por ejemplo, **help(print)**.

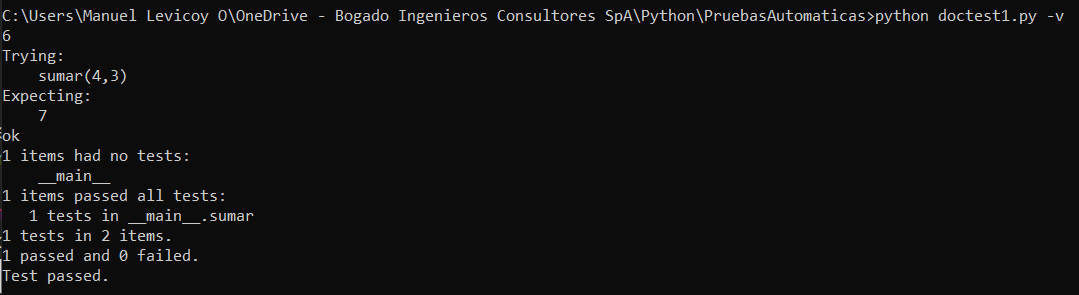


* Pydoc: Genera documentación automática desde la consola o terminal de comandos.
  + En Windows hay que ir a la ruta donde se encuentra la librería pydoc.py.
  + Se puede usar para acceder a cualquier documentación, por ejemplo, **pydoc len**.
  + Para generar un fichero HTML se hace con **pydoc -w <ruta del archivo py>**

1. **Pruebas automáticas**

* Doctest: Genera pruebas dentro de la documentación. Ejemplo,





Donde:

* + Se documenta una función.
  + Dentro de la documentación se realiza la prueba comenzando con 3 signos menor que y el resultado de la prueba abajo.



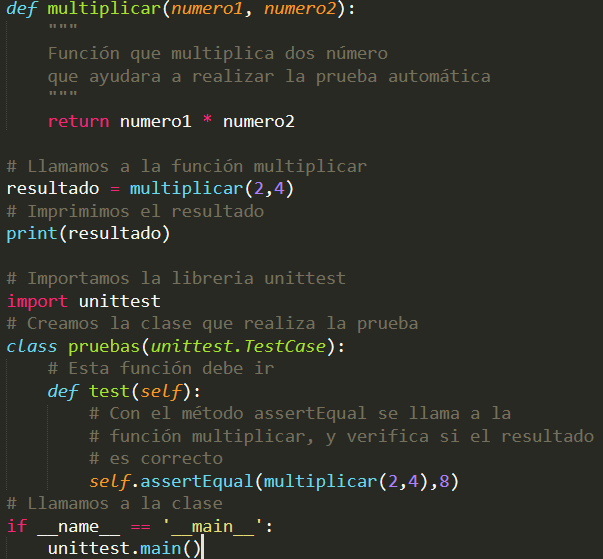
* + Para utilizar la función **doctest** hay que incluir al final,

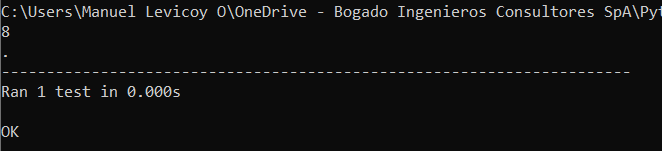


* + Desde la consola se ejecuta con **-v** para que las pruebas se realicen,



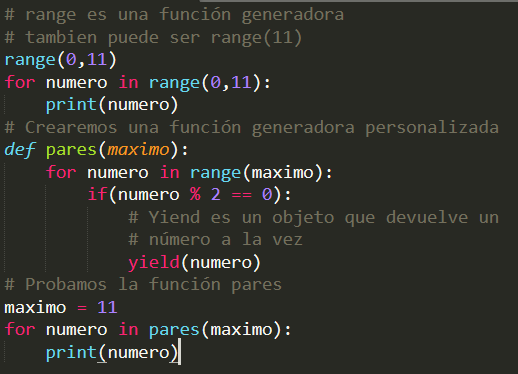
* Unittest: Sirve para crear pruebas dentro del propio código.



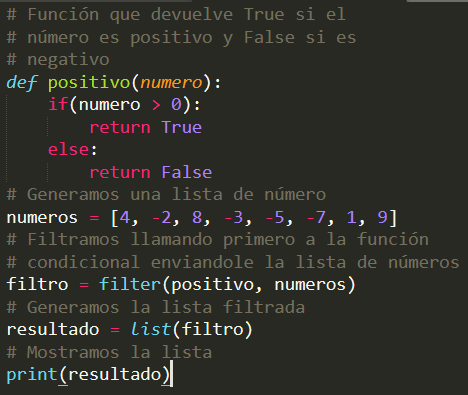


1. **Funciones avanzadas**

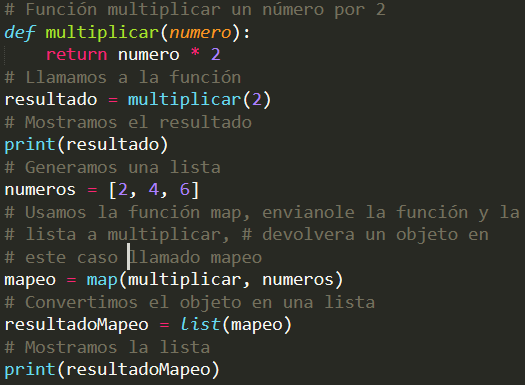
* Funciones generadoras: Genera valores sobre la marcha, cada vez que se le pide uno lo entrega. Por ejemplo,

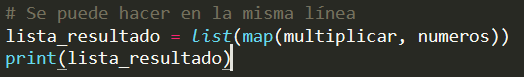


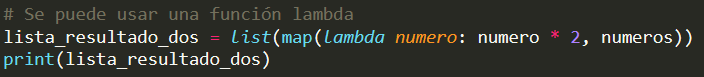
* Filter: Filtra resultados a partir de una lista y una función condicional. Y el resultado es otra lista con los elementos filtrados. Por ejemplo,



* Map: Aplica una función a una lista. Por ejemplo,



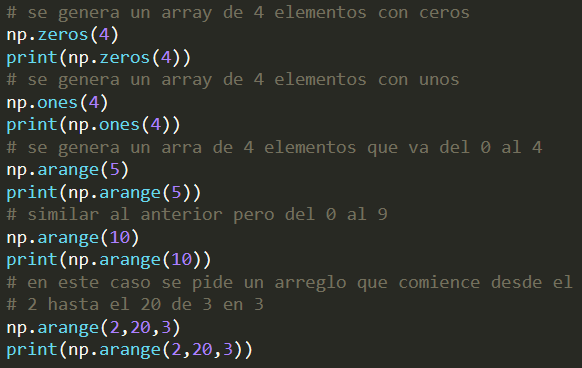


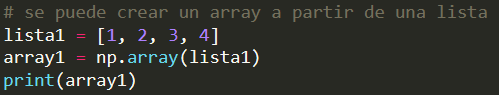


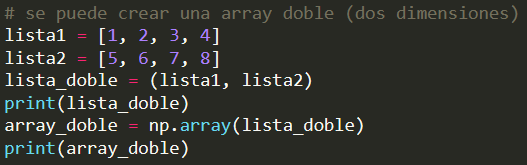
1. **Módulo numpy**

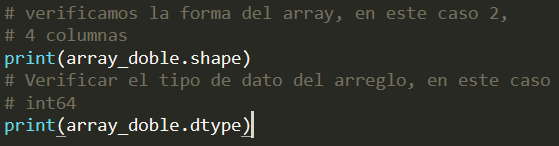
* Creando arrays:



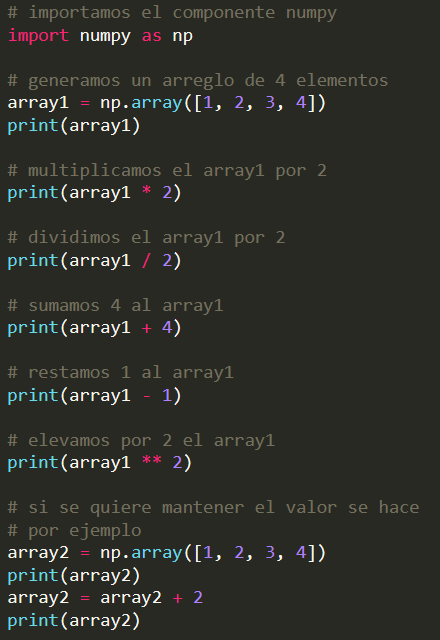


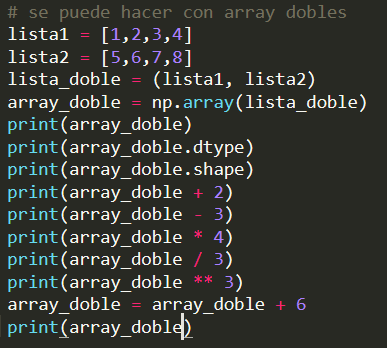






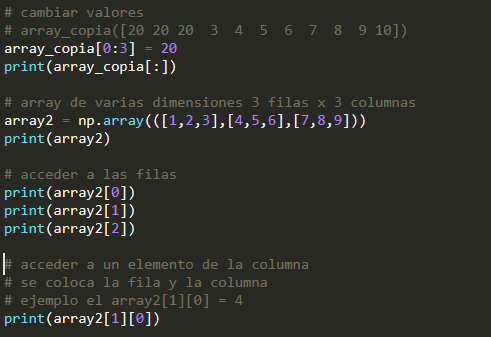
* Operaciones con array: (+, -, \*, /, \*\*)



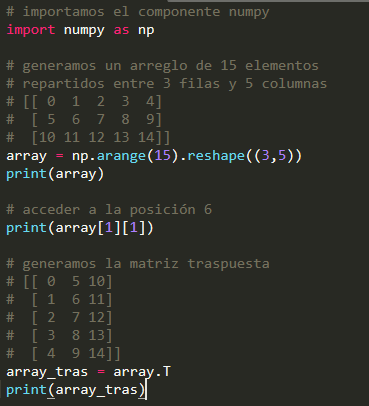


* Indexación:

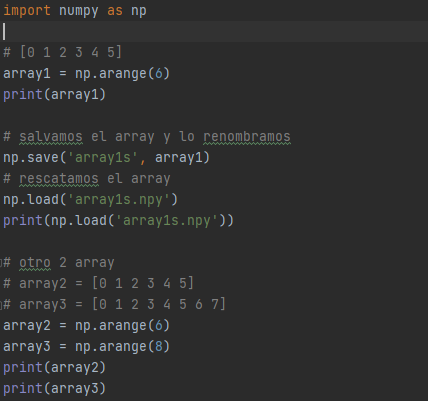


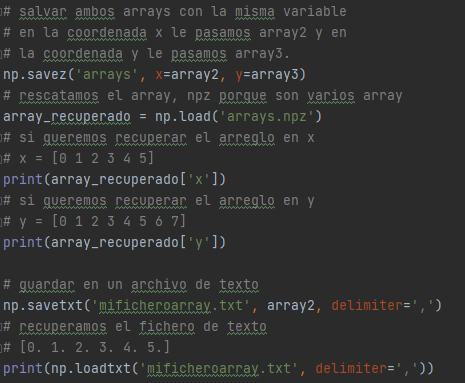


* Matrices traspuestas: permite cambiar ordenadamente las filas por las columnas.

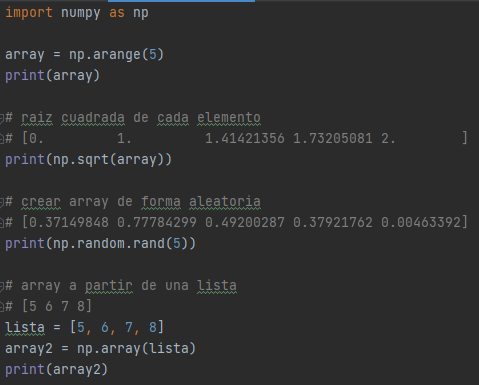


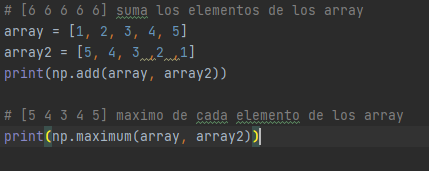
* Entrada y salida con array:



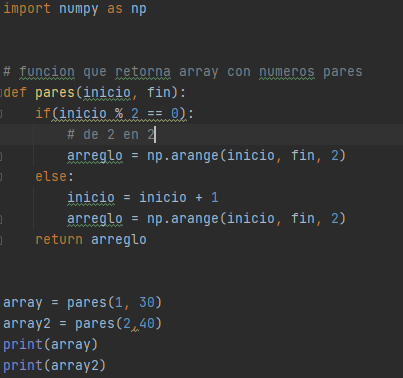


* Funciones con arreglos:

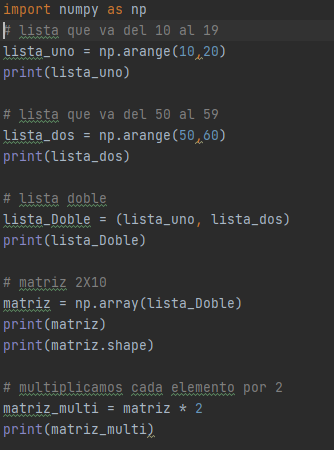




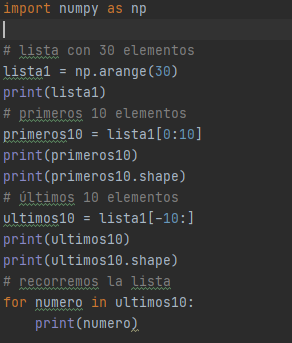
Ejemplo 1: Función que retorna un array con números pares de 2 en 2.



Ejemplo 2:

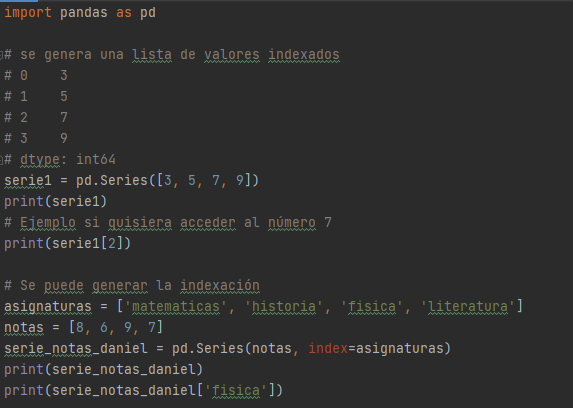


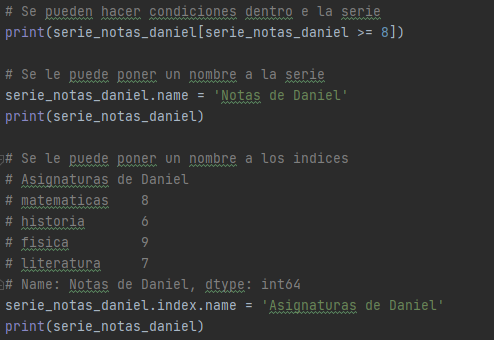
Ejemplo 3:

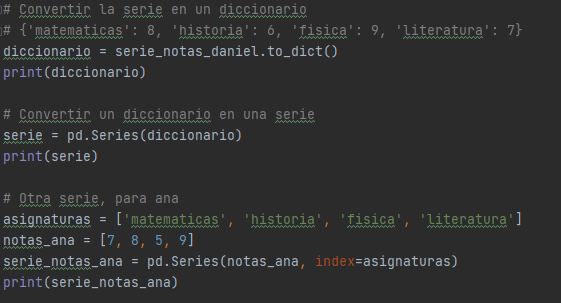


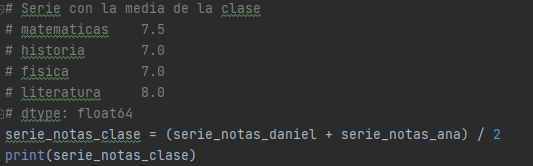
1. **Módulo pandas**

* Series: es una matriz unidimensional que contiene datos de cualquier tipo.

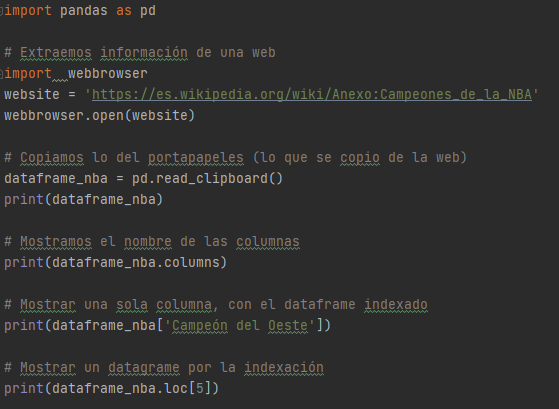


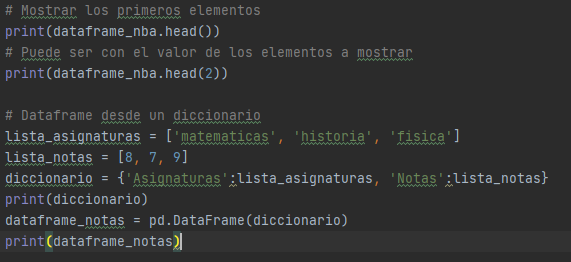






* DataFrames: estructura bidimensional, es decir, en filas y columnas.



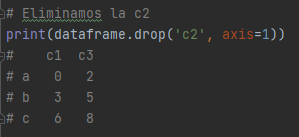


* Índices:

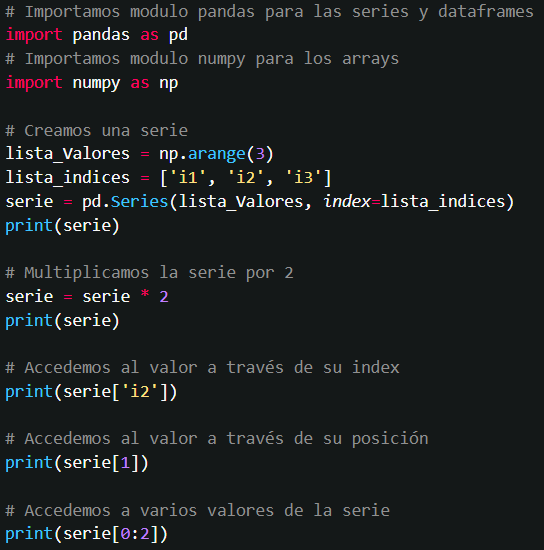


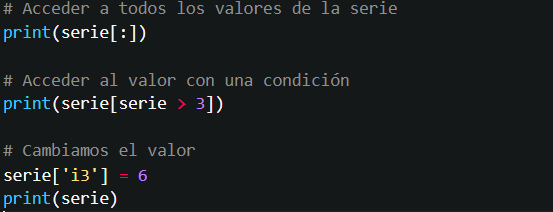
* Eliminar elementos:



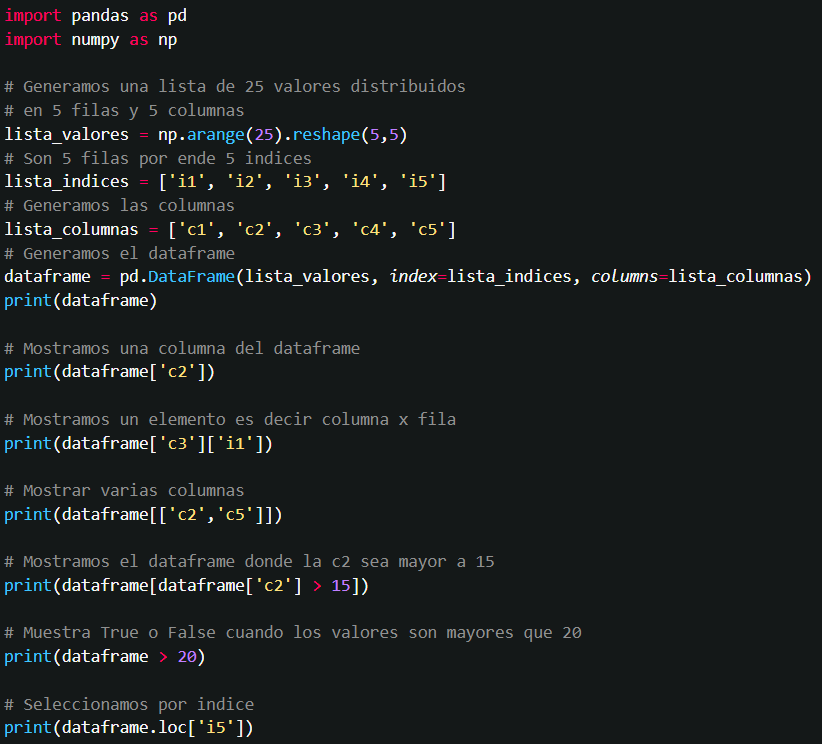


* Seleccionar datos en serie:

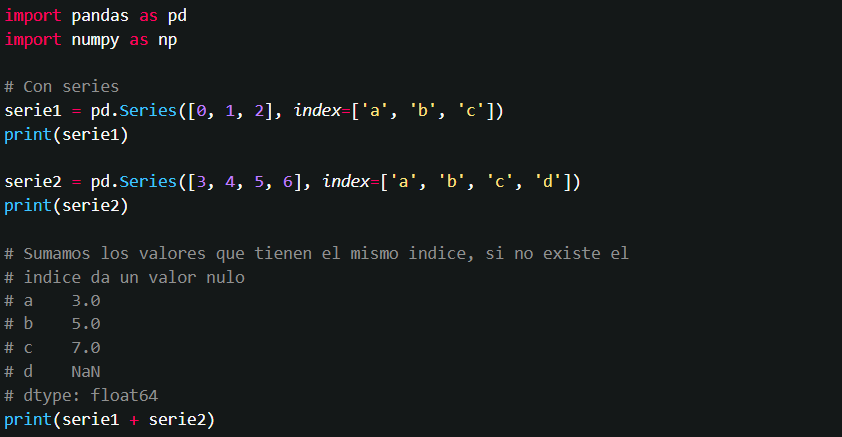


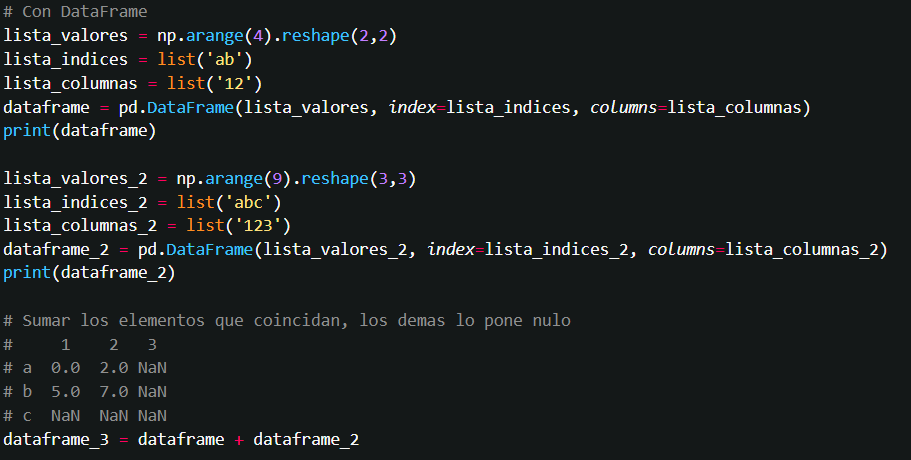


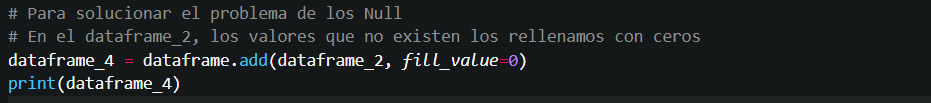
* Seleccionar datos en DataFrame:



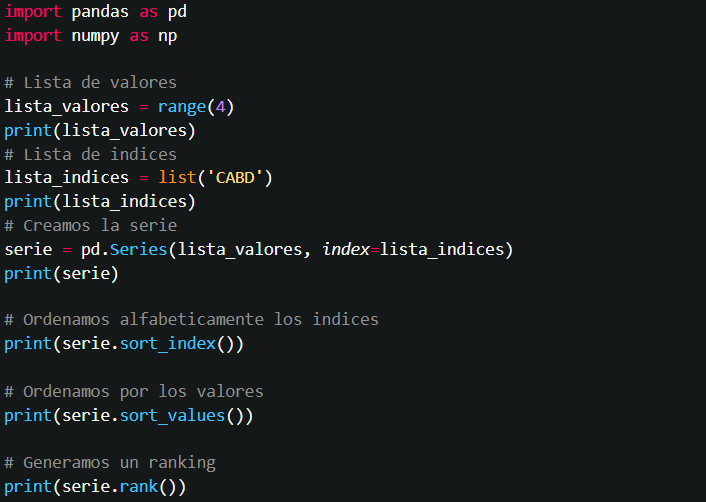
* Operaciones:

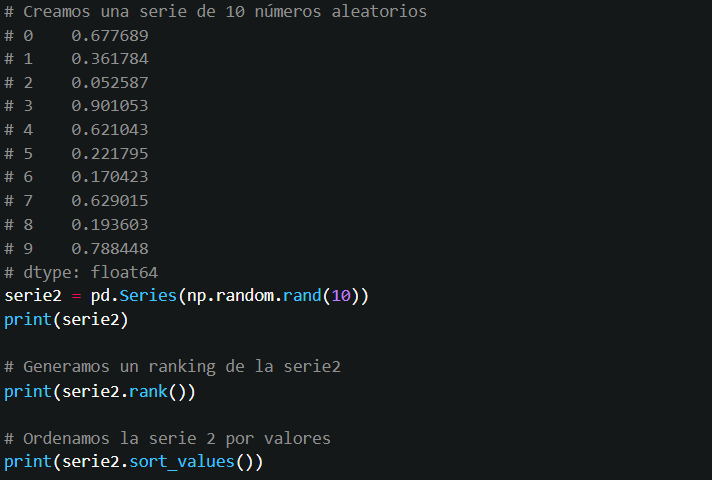




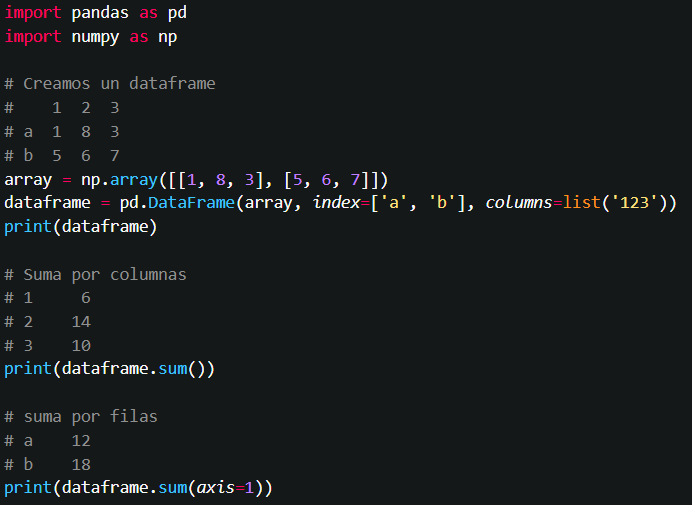


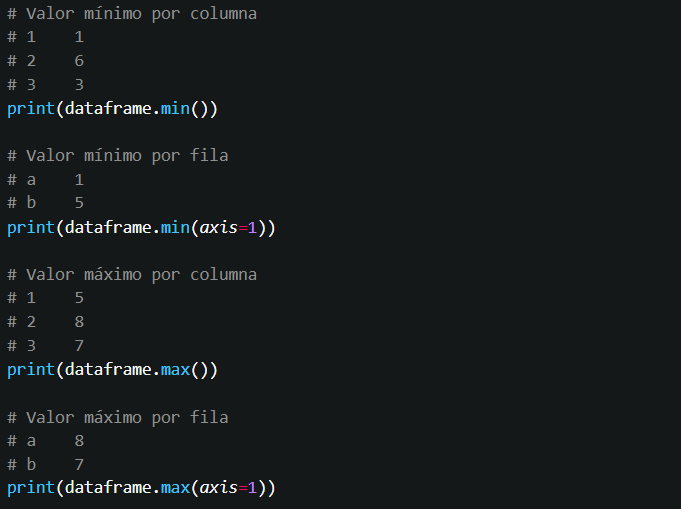
* Ordenación y clasificación:

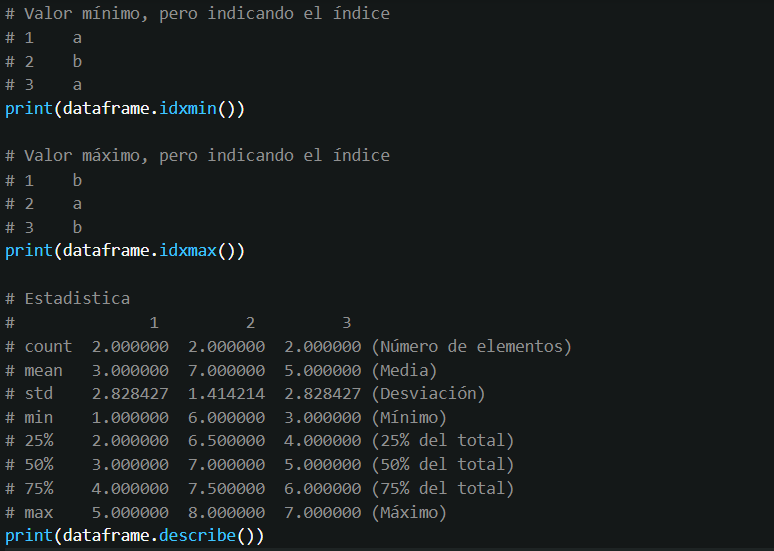




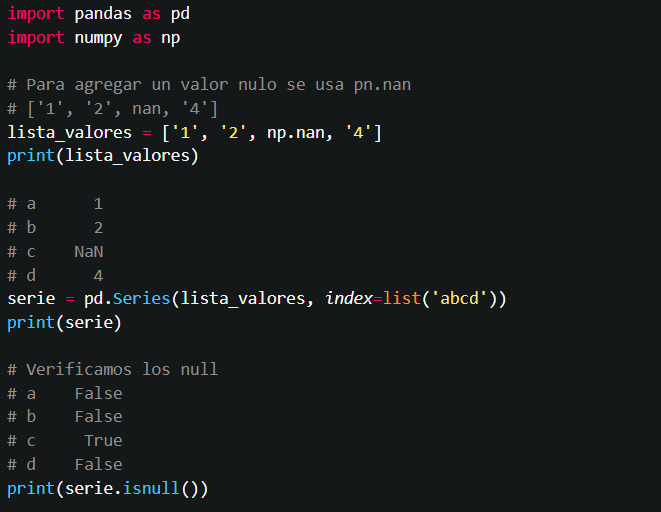
* Estadísticas:

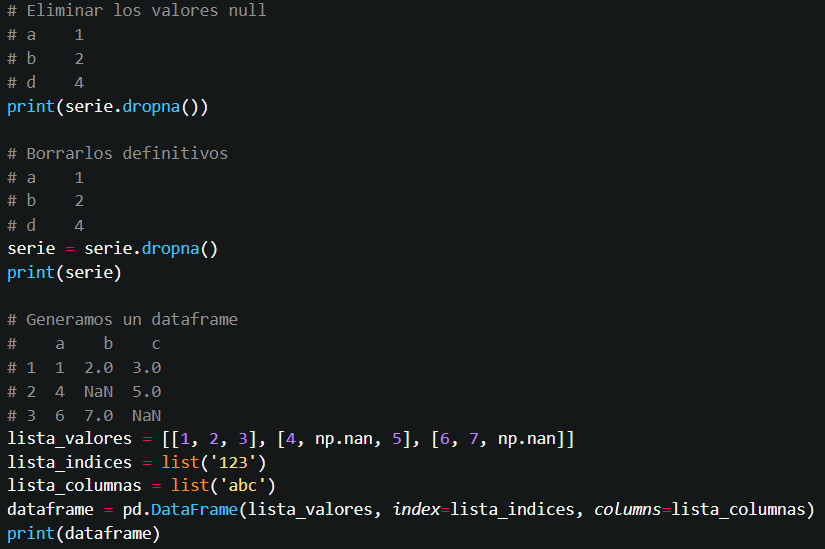


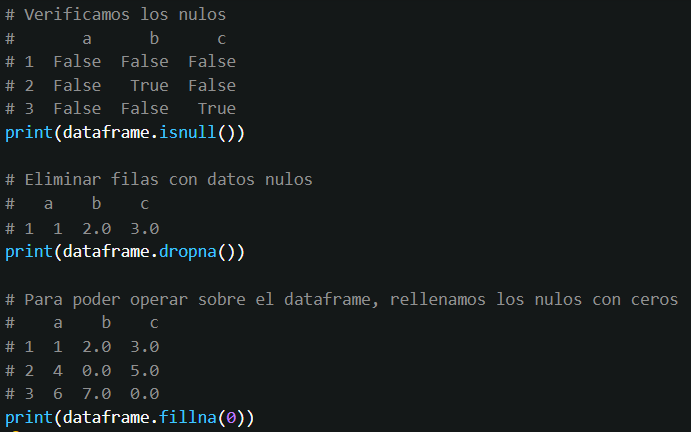




* Valores nulos:

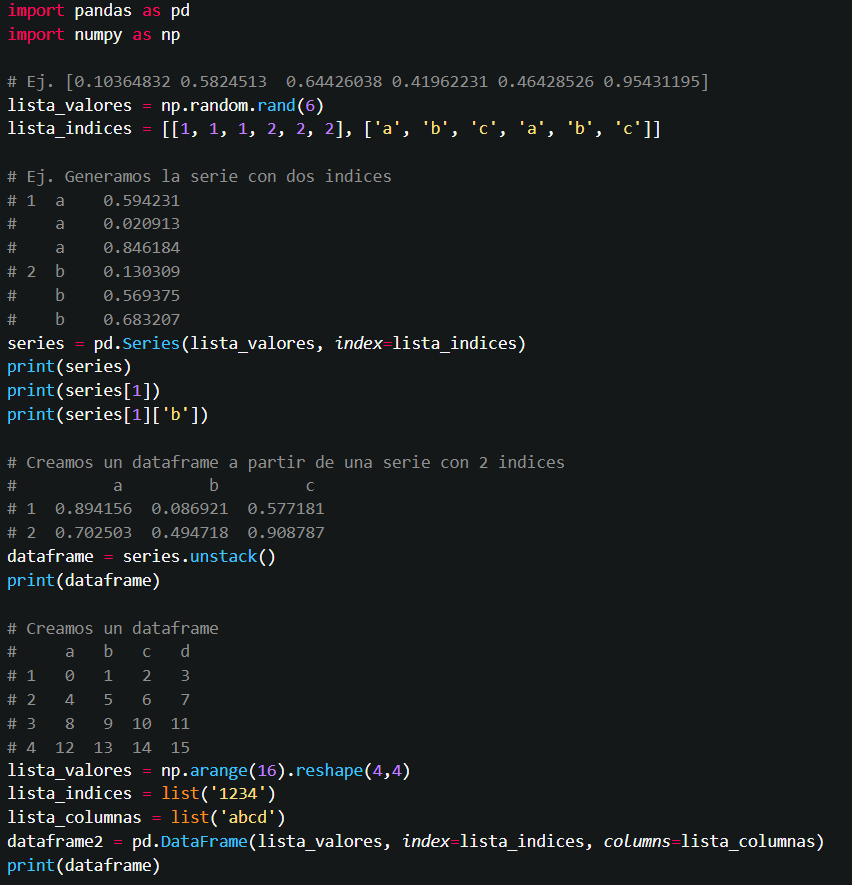


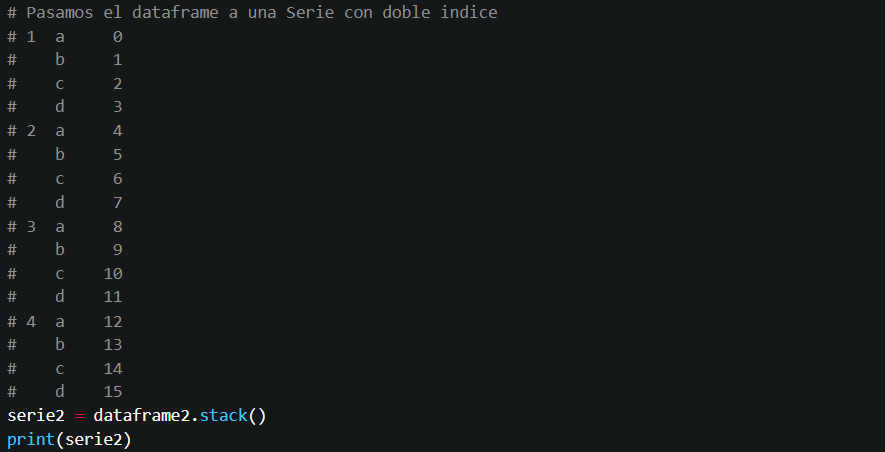




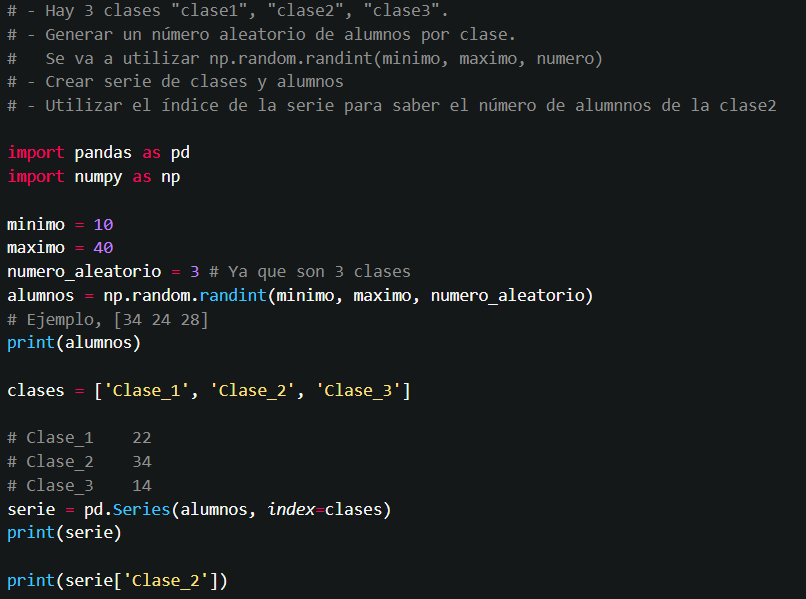


* Jerarquía de índices:

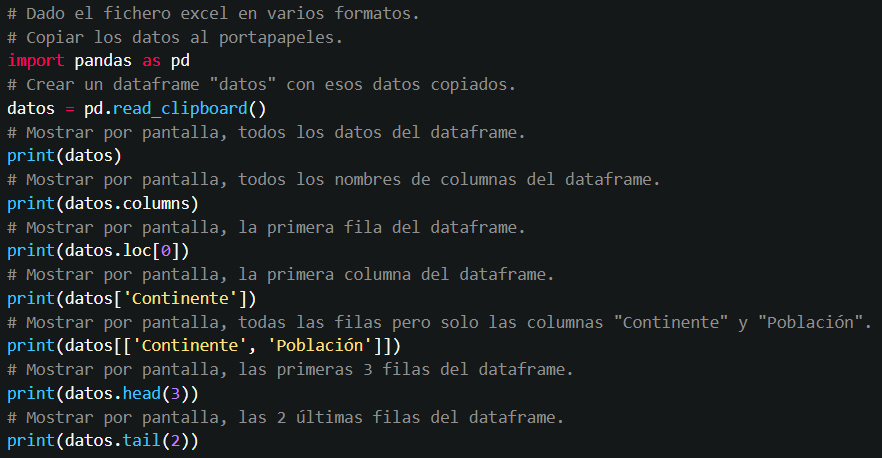




* Ejercicio 1:



* Ejercicio 2:

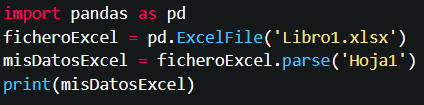


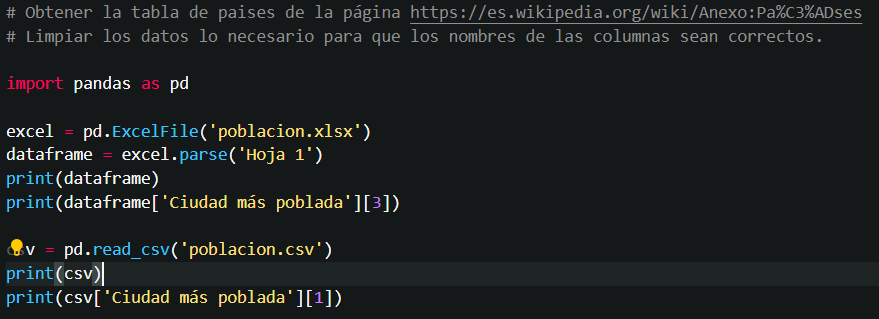
1. **HTML**

* HTML:



* Excel:





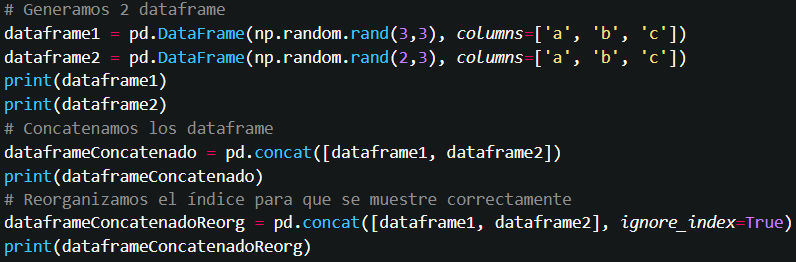
1. **Tratamiento de datos con DataFrame**

* Unión de dataframes:

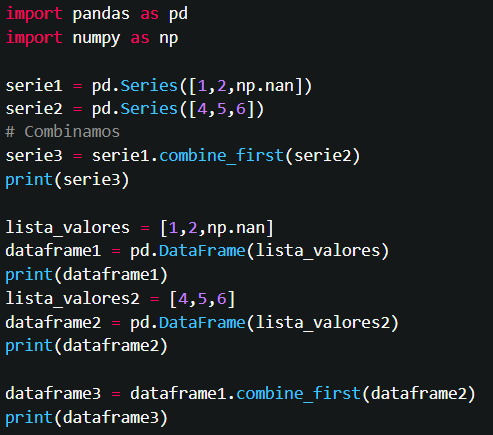


* Concatenación de datos:

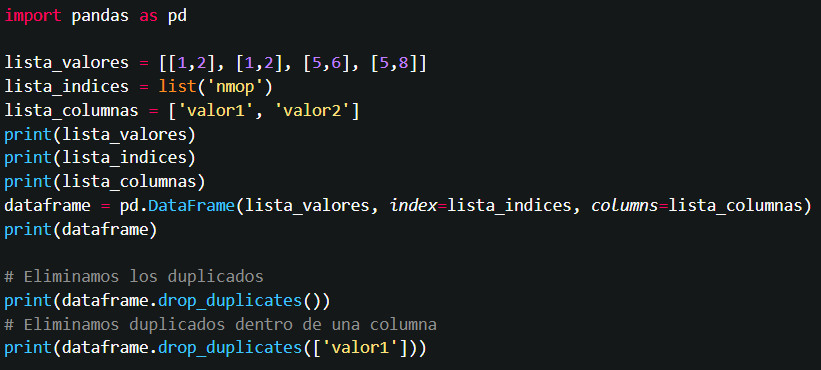




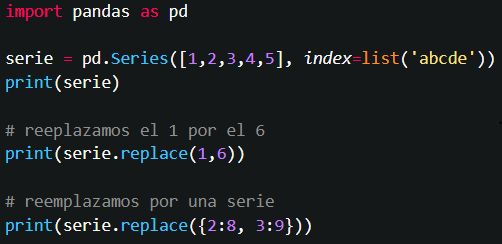
* Combinar Series y Dataframes:



* Duplicado Dataframe:



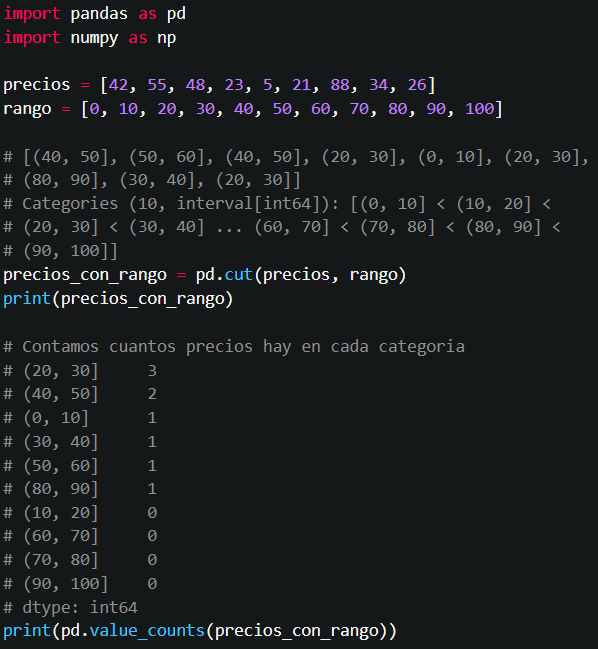
* Reemplazar datos en serie:



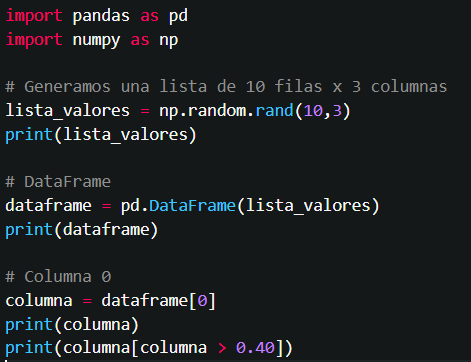
* Renombrar índices:



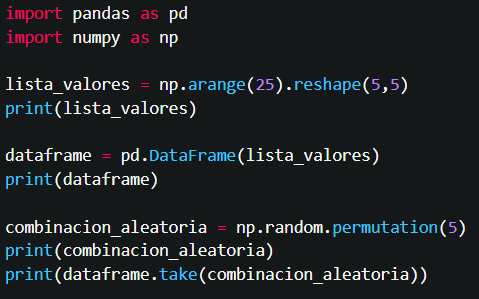
* Agrupar datos en categorías:



* Filtrar datos en DataFrames:



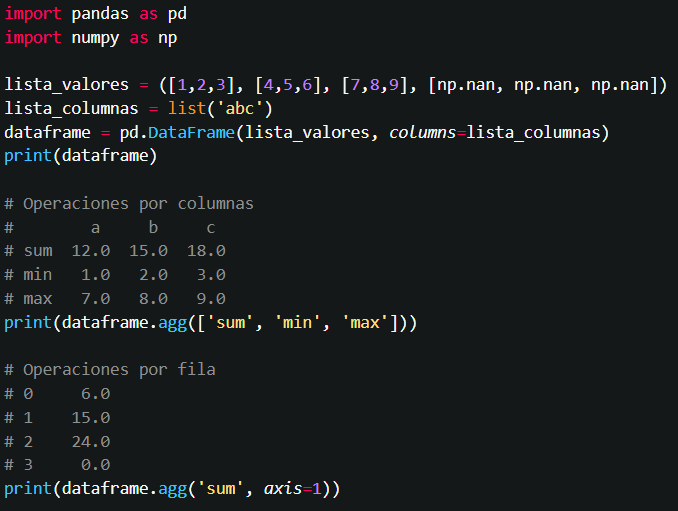
* Combinación de elementos:



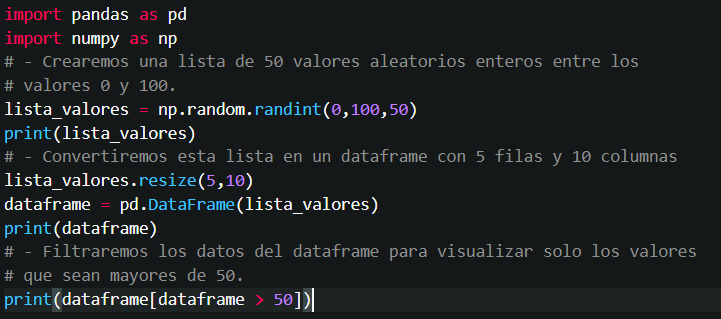
* Agrupación de Dataframe:



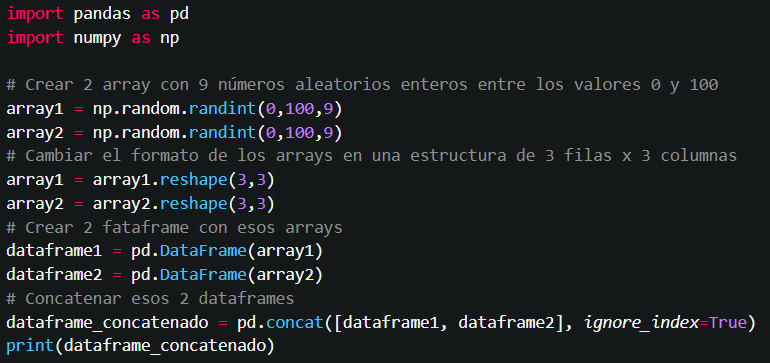
* Agregación DataFrame:



* Ejemplos 1:



* Ejemplo 2:

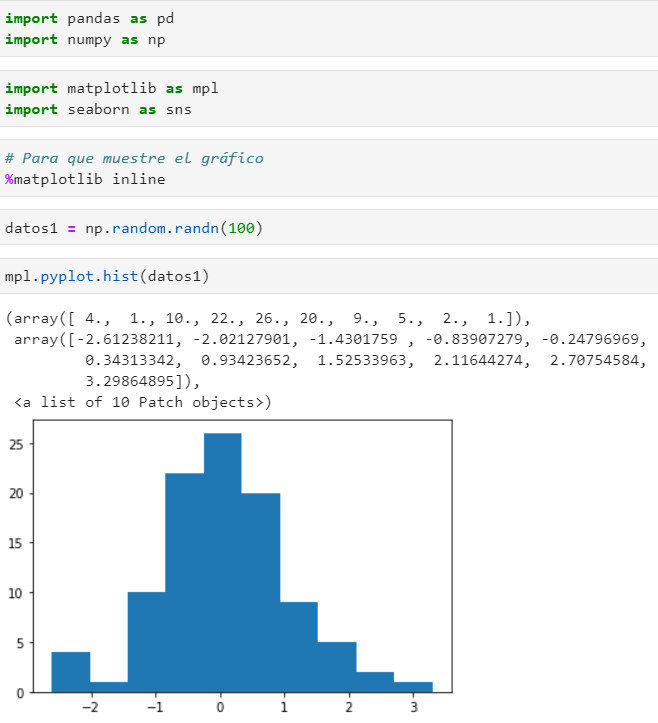


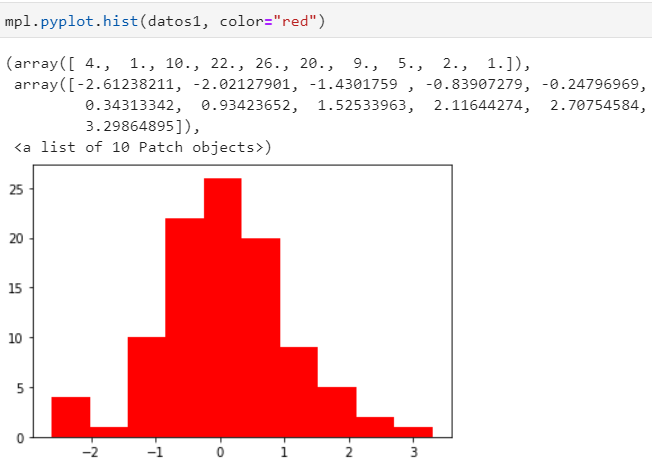
1. **Módulo seaborn y matplotlib**

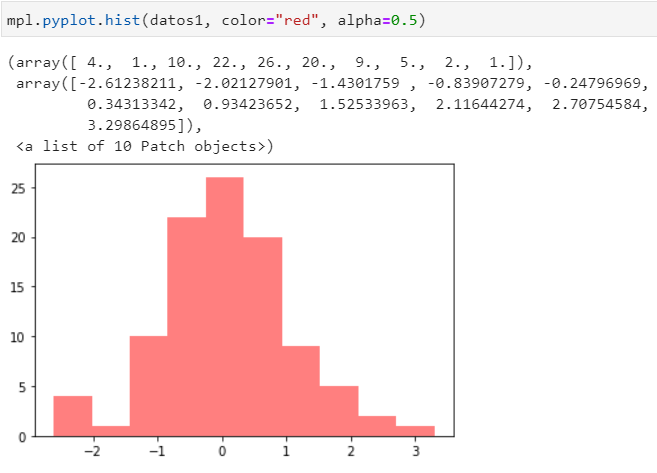
* Sirve para hacer gráficos estadísticos.
* Instalación:



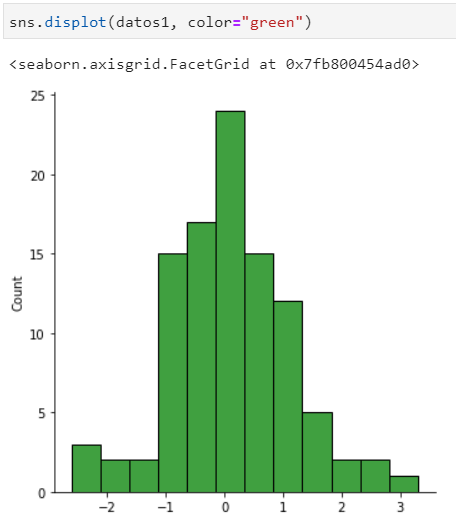
* Historigramas: Representación gráfica de una variable o unos datos en forma de barras. Donde la superficie de cada barra es proporcional a la frecuencia de los valores representados.

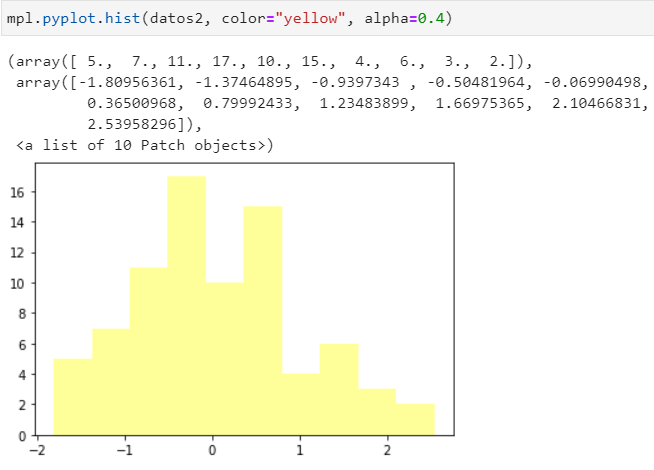






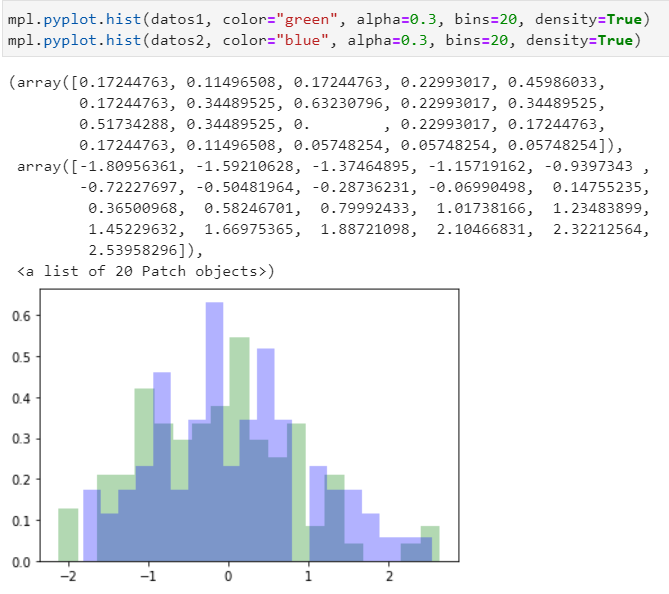
Ahora también se puede utilizar el módulo gráfico sns.

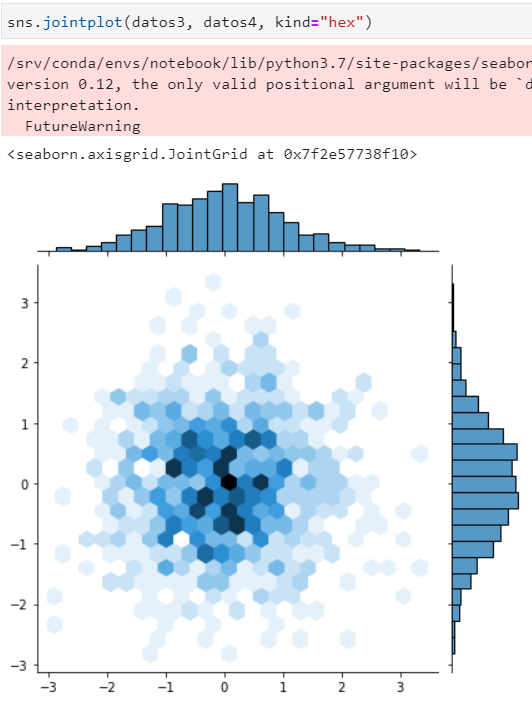




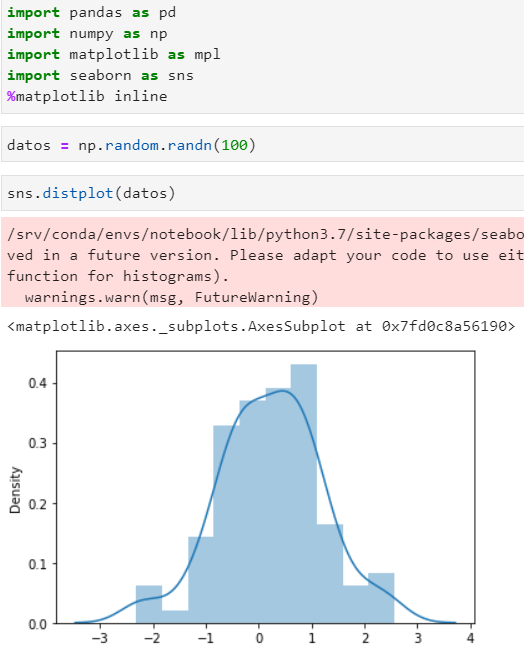
Generamos 2 gráficos:



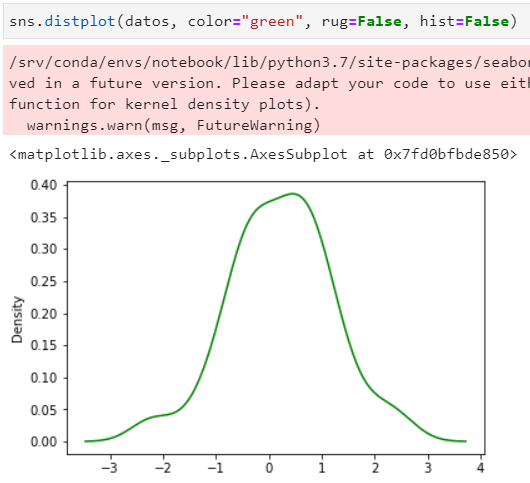




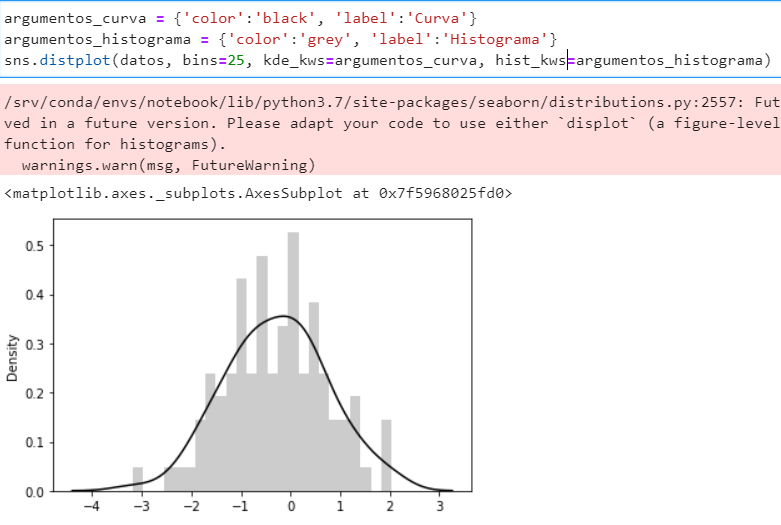
* Distribuciones:
  + Combinar estilos:



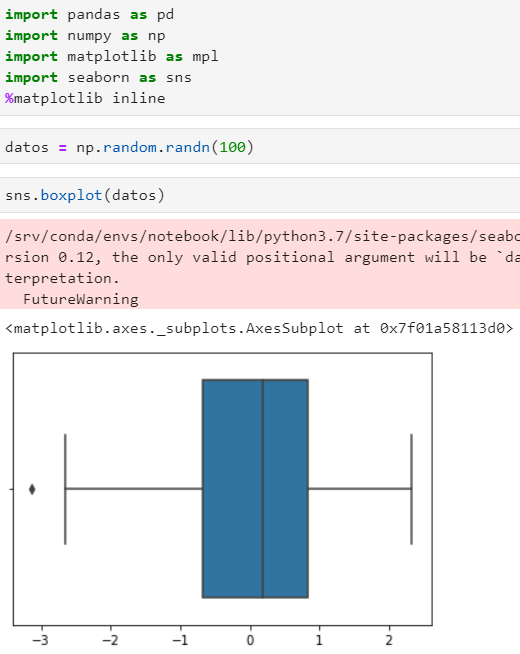
Eliminamos las barras:



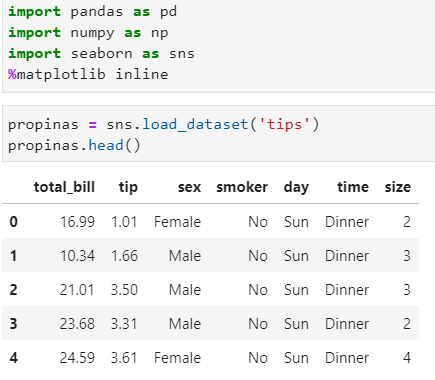
Color a la curva y color a las barras:

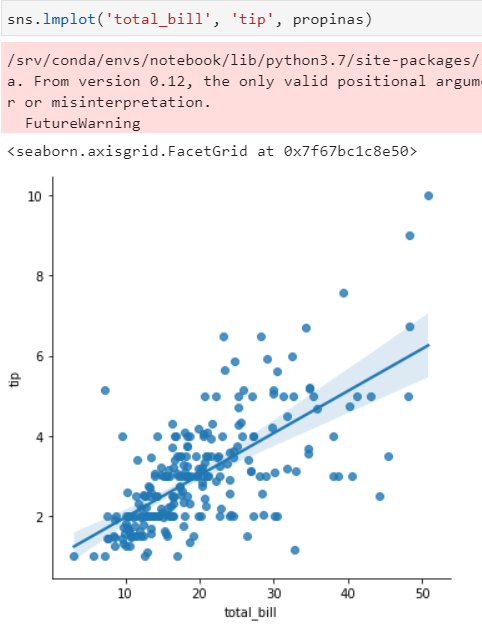


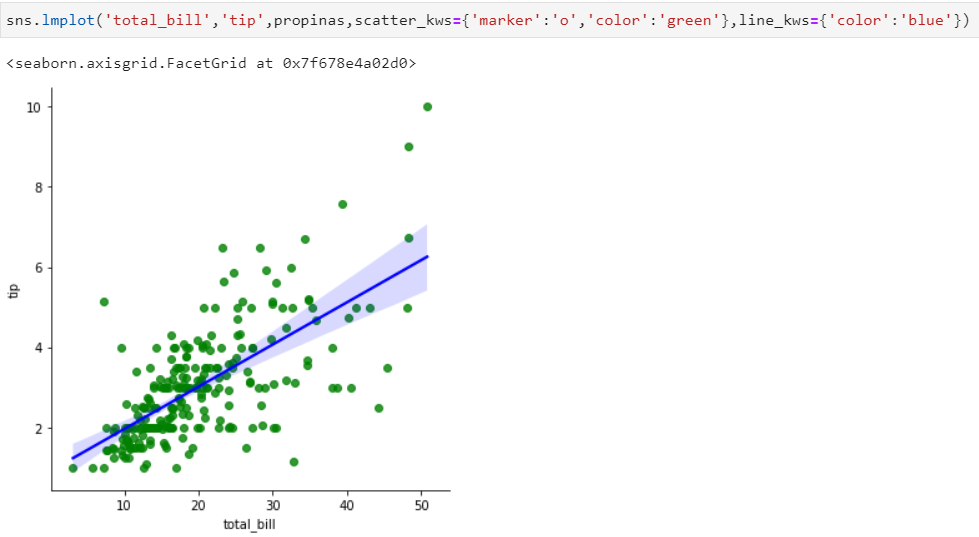
* Gráficos de caja:
  + Sirve para representar gráficamente una serie de datos numéricos a través de sus cuartiles.

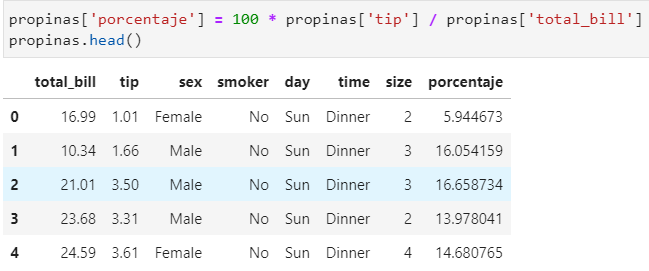


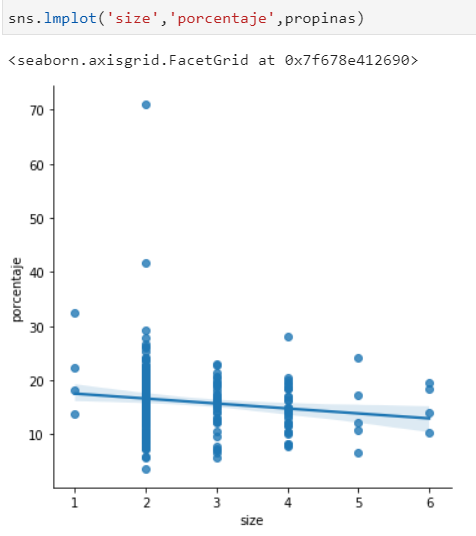
* Regresiones lineales: proceso estadístico para estimar las relaciones entre variables, es decir, entender como varia el valor de una variable en función del valor de otras variables.

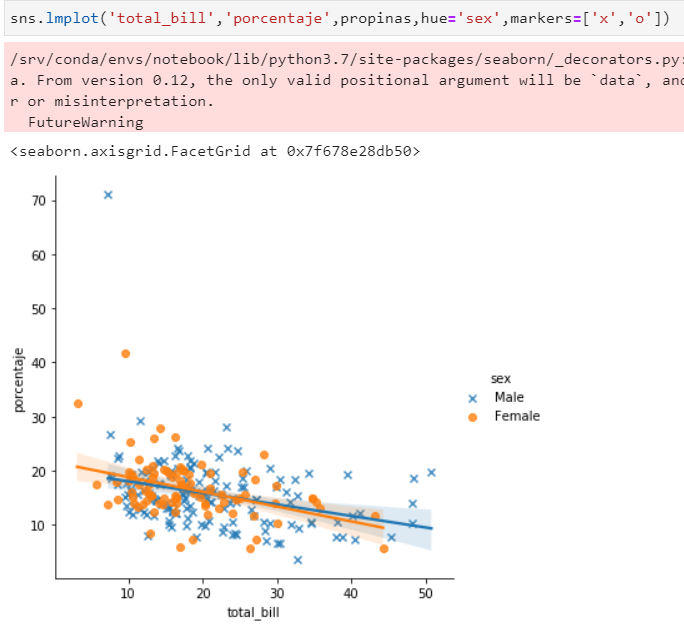






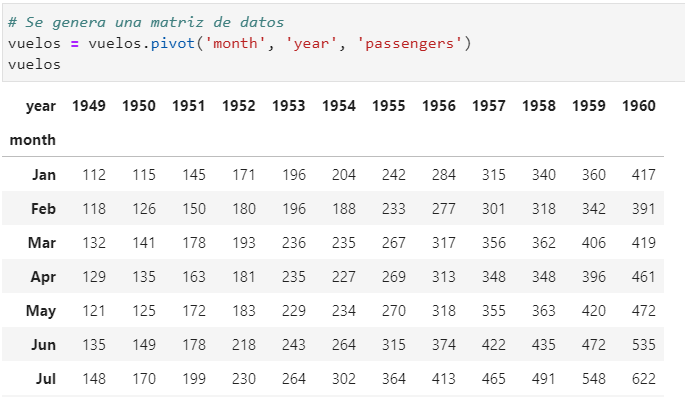


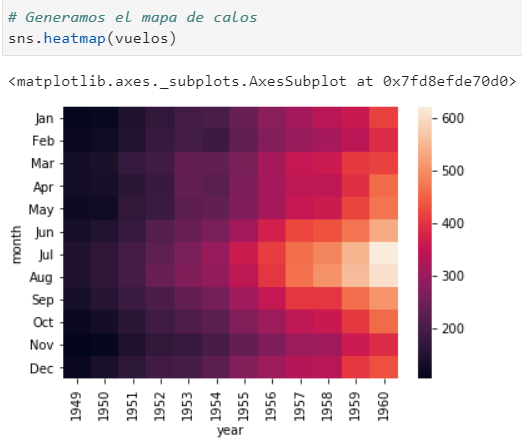


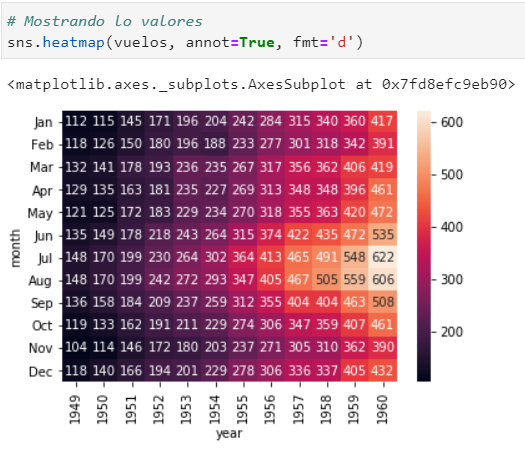


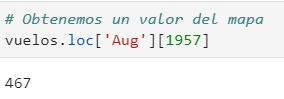
* Mapas de calor: Un mapa de calor es una representación gráfica de datos, donde los valores individuales contenidos en una matriz se representan como colores.

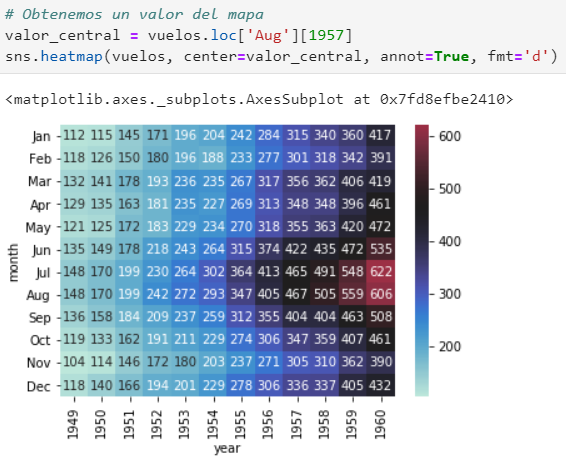


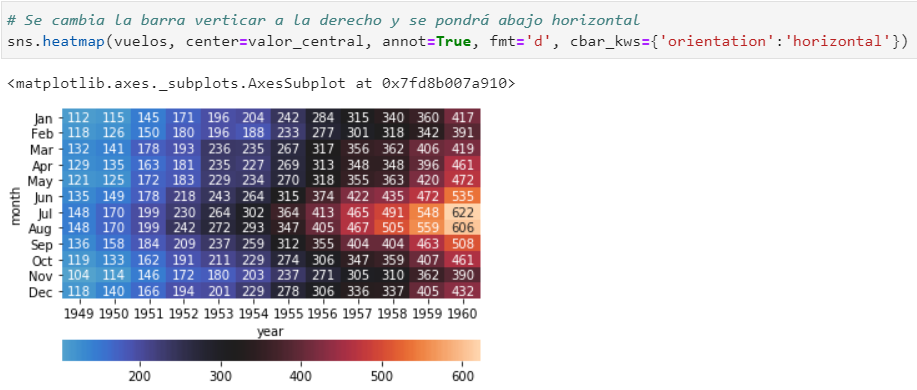




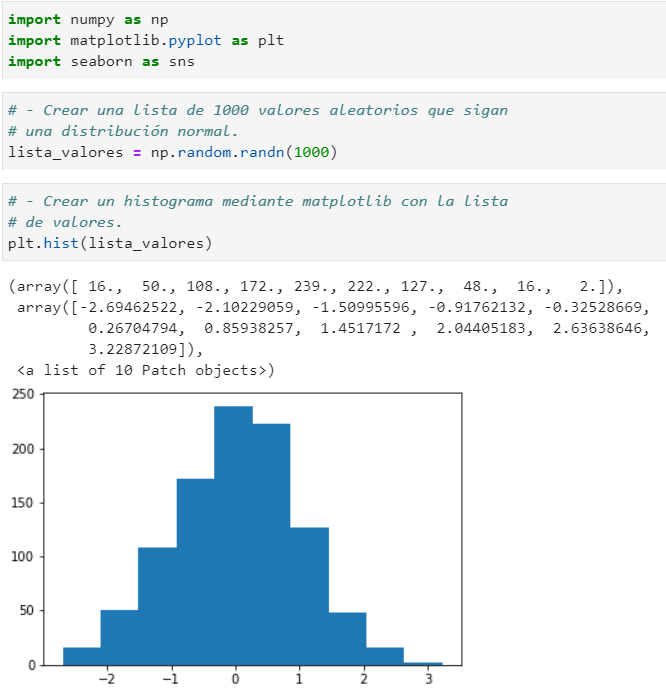


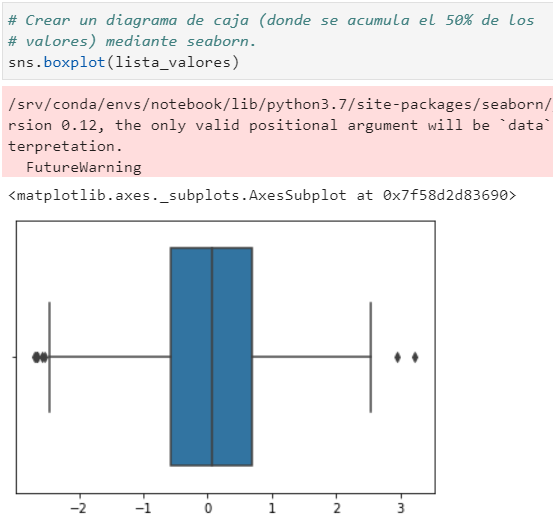






* Ejemplo 1:





* Ejemplo 2:

